# UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFOMÁTICA

Departamento de Ingeniería del Software e Inteligencia Artificial


## TESIS DOCTORAL

**Análisis, optimización, mejora y aplicación del análisis de dependencias**

**Analyzing, enhancing, optimizing and applying dependency analysis**

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Miguel Ballesteros Martínez

Directores

Virginia Francisco Gilmartín
Pablo Gervás Gómez-Navarro

**Madrid, 2013**

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA
Departamento de Ingeniería del Software e Inteligencia Artificial

# ANÁLISIS, OPTIMIZACIÓN, MEJORA y APLICACIÓN DEL ANÁLISIS DE DEPENDENCIAS

## ANALYZING, ENHANCING, OPTIMIZING AND APPLYING DEPENDENCY ANALYSIS


**MEMORIA PARA OPTAR AL GRADO DE DOCTOR PRESENTADA POR**

**Miguel Ballesteros Martínez**

Bajo la dirección de los Doctores

**Dra. Virginia Francisco Gilmartín**

**Dr. Pablo Gervás Gómez–Navarro**

Madrid 2012

# Análisis, Optimización, Mejora y Aplicación del Análisis de Dependencias


**Tesis Doctoral**

Presentada por
**Miguel Ballesteros Martínez**

Bajo la dirección de los Doctores

**Dra. Virginia Francisco Gilmartín**

**Dr. Pablo Gervás Gómez–Navarro**

Facultad de Informática
Universidad Complutense de Madrid
Madrid 2012

# Analyzing, Enhancing, Optimizing and Applying Dependency Analysis


**Ph.D. Thesis**

Presented by
**Miguel Ballesteros Martínez**

under the supervision of

**Virginia Francisco Gilmartín. Ph.D**

**Pablo Gervás Gómez–Navarro. Ph.D**

Facultad de Informática
Universidad Complutense de Madrid
Madrid 2012

## ACKNOWLEDGEMENTS

*Enjoy the little things,*
*for one day you may look back,*
*and realize they were the big things.*
*Robert Brault.*

23:52, 24th September, 2012, Madrid. Tomorrow I will deposit the thesis in the Department, and here I am willing to write this acknowledgements chapter which is the one that most people will read first. I feel that I would like to thank all the people that participate in a way or another in this thesis.

I naturally start with my advisors. *Virginia Francisco*, it can be summarized in two simple words: *always available*. She taught me how to write a paper by giving me an infinite number of revisions and suggestions with an infinite patience. The Gmail instant chat service is probably considering whether charging us. *Pablo Gervás*, my senior supervisor, he is like my academic *father*, an expert caring about what is going on, not only in the academic part of my life and always giving me the support, in all senses, that I needed to carry out this work. Among Pablo and Virginia, this work would have been impossible without the action of *Jesús* (or Χεσους Ερρεας) who introduced me in the world of syntax parsing technologies, he was, and he is, more than an advisor a friend, always sincere about my "enthusiastic" ideas and my perspectives.

The people from the Natural Interaction Based on Language Group (NIL) have had an important impact in my work. Specially, *Alberto* who had the role of my advisor more than once, this thesis is at least one fourth larger than planned and he is the one to blame for it. *Susana*, who listened very patiently to me in one of my worst moments and I will be always grateful for it, and thanks! for proofreading the Spanish version of the thesis. *Raquel*, it is amazing how someone can be always reachable and ready to help with a smile, it is true, you can find people like this. *Gonzalo* who was my professor in one of my earliest stages when I was starting my career and I could say that without him I would not have become the Computer Scientist that I am right now. *Jorge*, *Laura* and *Carlos*, they defended their theses when I was starting mine and they served me as a guideline about what I was expecting to achieve one day. The NIL group is hosted in the Department of Software Engineering and Artificial Intelligence, I would like to thank all the people from the department, from the head, thanks! *Luis*

for all the paper work, and your availability, through all the people, giving special emphasis to my office co-workers: *Dani*, my lunch mate, *Iván* and *Kiko*.

This acknowledgements chapter has an important main character: *Joakim Nivre*, *Joakim'*s work inspired me at the beginning of my PhD and turned out that I had the opportunity to work with him in Uppsala University by the end of 2011 and the beginning of 2012. In Sweden, he introduced me in the world of Machine Learning and Parsing Technologies in a way that I would never had dreamed, his extensive knowledge, his dedication about my ideas and the time he spent in me is something that I will never forget. I could say that I was feeling as Doctor Watson in a *Sherlock Holmes* story. *Interesting, though elementary, my dear Watson*, this is something that I never heard from him but I felt in the same way as Watson more than once, in the best of the senses you could imagine: learning a lot and being able to work with the best solving crimes, ups!, in parsing technologies! *Joakim*, you are the kind of researcher that I would like to become one day.

Also in Uppsala there are very nice people working in *Datorlingvistik*. *Evelina*, I hope you have learned from me at least one third of what I have learned from you. *Christian*, I really miss this Friday *drinks at 5* we used to have, I hope you are doing all right with your PhD and your career as *Basso*! *Jörg*, someone capable of sending emails at 8 am in the morning in the day in which I had to give a talk, just one week after he became a father for the third time, thanks! and thanks! again for proofreading. *Reut*, you were the one who was always smiling! *Matthias*, I am looking forward to come back, try your new beer and have an interesting chat. *Markus*, the kind of researcher you would like to expect in an audience if you are talking about transition-based dependency parsing, a *master*. *Mats*, we always had this controversial and politic conversations during the Swedish lunch time. Finally, *Johan* was not in Uppsala, but it was like he would, thanks! for your time, help and support.

When you are doing a PhD thesis you meet interesting people by travelling and attending conferences. *Hercules* and *Sumithra*, I had the chance to visit their department in Stockholm and the experience was just impressive. *Ryan*, who helped Joakim and I in our experiments, he was always there when I was wondering about something in which he could help, another *master*. *Jennifer*, thanks! for proofreading and for your invaluable comments. *Carlos*, I had the chance to publish a paper with one of the best in parsing from my own country. *Bernd* and *Xavier*, two that also belong to the kind of researcher that I would like to become in the future. And *Iria*, I believe that you know the important role you had, and I cannot be more thankful for it.

My family was always there supporting me. *Guillermo*, my brother, who is the light of the lighthouse that guided me several times in these difficult stormy nights, *grazie Guille*. My father *Jesús*, he is always supporting me

and finding something to laugh about, he has always a very good advice for every situation, I do not know how he manages. My mother *Alicia* who is an expert listener and she care (much) more about me than I could ever imagine. And also, my aunt *Gabi*, my uncle *Jorge* and my little cousin *Lucía*, thank you so much, I feel that you will always be there if I need it, I just hope you feel in the same way with me.

Of course my friends had a role: *Monique*, thanks for teaching me your *secret*, *Freckles*, you are the best. *Sergio*, you were at the other side of the very first door I knocked when I was disoriented, I believe that you know why. *Victor*, it is amazing how two friends can know each other so much, saying everything without saying nothing. *María*, thank you for letting me become who I am right now, I grew by your side and I spent one of the most important parts of my life just with you, you will always be in me. *Juan and Abel*, my challenging friends, always trying to find something new and different in order to have a good evening together. *Martín and Celia*, thanks both a lot for the encouragement and believing in me. And, *Jose*, my old co-worker and my running mate now, thanks!.

*Fede, Gonzo, Alberto, Pablo, Lolo, Elías, Álvaro, David, Fabio, Lara and company*: we were conquerors, we conquered Europe, from the west to the remotest corners of the east, even the north of Africa. One of the things that I regret most about the last years is that I missed most of the last battles. I will definitely compensate it.

Finally, it is worth mentioning that this thesis has been funded by the Spanish Ministry of Education and Science (TIN2009-14659-C03-01 Project) and the Universidad Complutense de Madrid.

*Miguel Ballesteros Martínez*

**ABSTRACT**

Statistical dependency parsing accuracy has been improved substantially during the last years. One of the main reasons is the inclusion of data-driven (or machine learning) based methods. Machine learning allows the development of parsers for every language that has an adequate training corpus without requiring a great effort. MaltParser is one of such systems.

In the present thesis we have used state of the art systems (mainly MaltParser), to show some contributions in four different areas inherently related to natural language processing (NLP) and dependency parsing: (i) We studied the parsing problem demonstrating the homogeneity of the performance and showing interesting contributions about sentence length, corpora size and how we normally evaluate the parsers. (ii) We have also tried some ways of improving the parsing accuracy by modifying the flow of analysis, parsing some segments of the sentences separately by finally constructing a parsing combination problem. We also studied the modification of the internal behavior of the parsers focusing on the root of dependency structures, which is an important part of what a dependency parser parses and worth studying. (iii) We have researched automatic feature selection and parsing optimization for transition based parsers which we consider an important problem and something that definitely needs to be done in dependency parsing in order to solve parsing problems in a more successful way. And (iv) we have applied syntactic dependency structures and dependency parsing to solve some Natural Language Processing (NLP) problems such as text simplification and inferring the scope of negation cues.

Furthermore, the knowledge acquired when developing this thesis could be used to implement more robust dependency parsing–based applications in different NLP (or related) areas, as we demonstrate in the present thesis.

## RESUMEN

Los analizadores de dependencias estadísticos han sido mejorados en gran medida durante los últimos años. Esto ha sido posible gracias a los sistemas basados en aprendizaje automático que muestran una gran precisión. Estos sistemas permiten la generación de parsers para idiomas en los que se disponga de un corpus adecuado sin causar, para ello, un gran esfuerzo en el usuario final. MaltParser es uno de estos sistemas.

En esta tesis hemos usado sistemas del estado del arte, para mostrar una serie de contribuciones completamente relacionadas con el procesamiento de lenguaje natural (PLN) y análisis de dependencias: (i) Estudio del problema del análisis de dependencias demostrando la homogeneidad en la precisión y mostrando contribuciones interesantes sobre la longitud de las frases, el tamaño de los corpora de entrenamiento y como evaluamos los parsers. (ii) Hemos estudiado además algunas maneras de mejorar la precisión modificando el flujo de análisis de dos maneras distintas, analizando algunos segmentos de las frases de manera separada, y modificando el comportamiento interno de los algoritmos de parsing. (iii) Hemos investigado la selección automática de atributos para aprendizaje máquina para analizadores de dependencias basados en transiciones que consideramos un importante problema y algo que realmente es necesario resolver dado el estado de la cuestión, ya que además puede servir para resolver de mejor manera tareas relacionadas con el análisis de dependencias. (iv) Finalmente, hemos aplicado el análisis de dependencias para resolver algunos problemas, hoy en día importantes, para el procesamiento de lenguage natural (PLN) como son la simplificación de textos o la inferencia del alcance de señales de negación.

Por último, añadir que el conocimiento adquirido en la realización de esta tesis puede usarse para implementar aplicaciones basadas en análisis de dependencias más robustas en PLN o en otras áreas relacionadas, como se demuestra a lo largo de la tesis.

# Contents

# List of Tables

# List of Figures

# Part I

# Analyzing, Enhancing, Optimizing and Applying Dependency Analysis

# Chapter 1

# Introduction

*Set your goals high, and don't stop till you get there.*
*Bo Jackson*

The performance of statistical parsers for natural language has improved tremendously during the last two decades, and there are now a number of different systems that can be used to develop parsers for new languages and applications. This includes constituency-based parsers like the Stanford Parser (Klein and Manning, 2002) and the Berkeley Parser (Petrov et al., 2006) as well as dependency parsers such as MaltParser (Nivre, Hall, and Nilsson, 2004) and MSTParser (McDonald, Crammer, and Pereira, 2005).

The dependency parsing community is important within the Natural Language Processing community (NLP), and the Computational Linguistics community (CL). We can observe how in the most recent big conferences, even in the year of this writing, there are several sessions dedicated to parsing topics in which researchers have the opportunity to show their contributions. We can find a clear example of this fact in the Shared Tasks of the 10th and 11th edition of the Conference of Computational Natural Language Learning (CoNLL). These Shared Tasks were completely based on Multilingual Dependency parsing (Buchholz and Marsi, 2006; Nivre et al., 2007). A set of different languages were involved and parsing performance from several perspectives was studied. Moreover, and to provide another example, in the most recent conference of the European Chapter for Computational Linguistics of 2012 (EACL) (Daelemans, Lapata, and Màrquez, 2012), the best paper award went to a dependency parsing paper (Luque et al., 2012), and there were several interesting works concerning this topic, such as (Bohnet and Kuhn, 2012), (Farkas, Vincze, and Schmid, 2012), (Tsarfaty, Nivre, and Andersson, 2012), (Gómez-Rodríguez and Fernández-González, 2012) and (Ballesteros and Nivre, 2012).

We believe that the Dependency Parsing Community is receiving a lot

of interest because there is still much room for improvement and there is the possibility of finding different ways of solving the related problems in a better way by coming up with novel approaches.

The dependency parsing application area is growing fast, and we certainly believe that in the future there will be several contributions in which researchers will make use of new improvements in dependency parsing to apply the usefulness that it provides to solve several natural language processing problems. For instance, we can already find some applications that use dependency parsing as a basis for their work, such as textual entailment recognition (Herrera, Peñas, and Verdejo, 2005), relation extraction (Culotta and Sorensen, 2004), machine translation resources (Tiedemann, 2012), question–answering systems (Cui et al., 2005), negation detection (Councill, McDonald, and Velikovich, 2010), synonym generation (Shinyama, Sekine, and Sudo, 2002), surface realization (Bohnet et al., 2011), lexical resource augmentation (Snow, Jurafsky, and Ng., 2005) or parsing social media (Foster et al., 2011).

In this thesis we had the idea and the opportunity of studying the parsing problem in depth, applying the parsers to several domains, topics and languages by acquiring expert knowledge that could be very useful for future purposes. This is why we studied the dependency parsing problem from four different perspectives:

1. How can we analyze this problem?
2. Can we enhance the accuracy by modifying the flow of analysis or the behavior of the parsers?
3. Can we automatically optimize a dependency parser?
4. Can we apply dependency structures to solve some interesting Natural Language Processing problems?

All the work included in this thesis is intrinsically connected, as it is practically impossible to solve one of the already mentioned problems without the notions acquired in another. In the present Chapter, we describe the problem of dependency linguistics and dependency parsing in a formal way. We also show the objectives and structure of the present thesis.

## 1.1 Dependency Linguistics and Dependency Parsing

The modern theory of dependency in language comes from Lucien Tesnière (1959) with the aim of building a grammar useful for teaching languages, however there were researchers who pointed out that it was already used during the Middle Ages (Covington, 1984). There were others in the 60s

and 70s that emphasized the idea of dependency in language, such as Hays (1964) or Robinson (1970). Moreover, the dependency theory from Igor Mel'čuk in the 80s is very relevant (1988), because he established the basis of what we nowadays understand as dependency linguistics in the context of the meaning text theory (MTT), especially for Slavic languages because they allow more "freedom" in the word order. During the last years dependency linguistics has had a very long tradition in descriptive linguistics, and more recently it is receiving a lot of interest in the computational linguistics community. As an example, Nivre (2005) argued that the latest boom that dependency grammar is having is due to the potential linguistic usefulness that dependency relations provide for several computational purposes.

In dependency parsing, the syntactic structure of language consists of different lexical items linked by binary asymmetric relations called dependencies. A dependency structure for a sentence is therefore a labeled directed graph, consisting of a set of nodes, labeled with words, and a set of directed arcs (or edges) that may be labeled (or not) with dependency types. Dependency parsing is mainly the syntactic parsing of natural language, based on syntactic dependency representations. Taking this into account, having a dependency parser we are therefore capable of building such dependency structure given an input sentence (Nivre, 2006; Kübler, McDonald, and Nivre, 2009).

During the following Subsections we summarize the definitions and principal concepts related to dependency linguistics and dependency parsing, as a way of introducing it to the reader. We show in bold the concepts that are important and relevant to understand what we explain in this thesis.

### 1.1.1 Dependency Structures

In a formal fashion, a **dependency graph** for a sentence (made up of tokens or wordforms) S = $w_1$,...,$w_n$ is a directed graph G = (V,A), where:

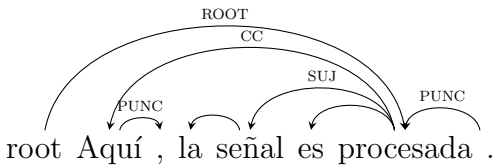
- V = $\{1, ....n\}$ is the set of **nodes**, representing tokens.
- A $\subseteq$ V x V is the set of **arcs**, representing dependencies.
- An arc i $\rightarrow$ j is a dependency with head $w_i$ and dependent $w_j$.
- An arc i $\rightarrow$ j may be labeled with a dependency type contained in the set of possible dependencies.

Dependency graphs are normally assumed to satisfy certain formal constraints, such as the single-head constraint, which forbids more than one incoming arc to a node, and the acyclicity constraint, ruling out cyclic graphs. Many dependency theories and annotation schemes further require that the graph should be a tree, with a unique root token on which all other tokens are transitively dependent, while other frameworks allow more than one token to be a root in the sense of not having any incoming arc.

Therefore, a dependency graph should respect the following:

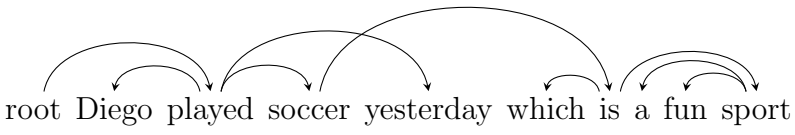
- The graph must be connected and acyclic.
- A node may have more than one incoming arc. However, sometimes it is required that a node cannot have more than one incoming arc.

We will refer to the dependency structures as **dependency trees** (or even trees for simplicity) in this manuscript. A dependency tree is a dependency graph with a single root node and there can not be any incoming arc to the root. In some corpora the dependency structures are *graphs* but for most of the purposes of the present thesis, the dependency structures are actually dependency trees. The Figure shown below is an example of a labeled dependency tree for the sentence *Aquí, la señal es procesada [Here, the signal is processed].*

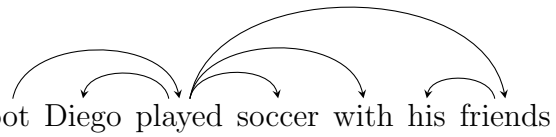



The dependency trees may be **projective** or **non-projective**. A dependency tree is projective if and only if, for every arc wi→wj, and every node wk between wi and wj (i ≤ k ≤ j) in the original order, there is a directed path from wi to wk (being a path, a set of arcs that connects several nodes).

The following dependency tree is therefore non-projective, because there is no direct path from *soccer* to *yesterday*:




and the following one is therefore projective, because there is a directed path between every two nodes in the dependency tree:



### 1.1.2 Dependency Parsing

The parsing problem consists in coming up with a system that is able to generate a dependency structure for an input sentence. Given a sentence $w_1 \dots w_n$, the goal of a dependency parser is to assign to this sentence a *dependency graph*, which is a directed graph $G = (V, A)$ where $V = \{1, \dots, n\}$ and $A \subseteq V \times V$.[1] As we mentioned before, such dependency analyses are normally required to satisfy the acyclicity constraint (i.e. that the graph cannot have cycles) and the single-head constraint (that a node cannot have more than one incoming arc).

In a formal fashion we can say that the goal is to maximize the score (being the score, the confidence of the system in the output) of the produced graph for a given sentence, as we show below:

- Input: S = $w_1$,...,$w_n$.
- Output: G* = argmax F(S,G) where G∈G(S).
  - F(S,G) = score of G for S.
  - G(S) = search space for S (being the search space, a set containing all the possible graphs for the given sentence).

There are several approaches that solve the problem of dependency parsing. Besides the old-fashioned rule-based parsers that are based on grammars and a complex set of rules. There are mainly two model families that are **data-driven** and based on **machine-learning**:[2]

- The **Graph–based family** (Eisner, 1996), that parameterizes parsing models by dependency arcs and tries to predict entire trees by giving (and ranking) weight to all the possible arcs for a given sentence. The most representative example for this technology is **MSTParser** (McDonald et al., 2005; McDonald, Lerman, and Pereira, 2006), but we can also find other systems such as Corton's (2006), Dreyer's (2006) or Carreras's (2007; 2008).
- The **Transition–based family** (Nivre, Hall, and Nilsson, 2004) (Yamada and Matsumoto, 2003), in which parsing models are parametrized by state transitions whose actions (transitions) manipulate input words and build dependency relations between them in a deterministic fashion. **MaltParser** (Nivre, Hall, and Nilsson, 2006) is the most representative parser for this technology. Johansson's (2006), Cheng's (2006) or Wu's (2006) systems follow this model as well.

[1] We ignore arc labels in this explanation for simplicity of presentation, but the arcs in the set $A$ may be labeled

[2] See Section 2.1, to find a complete description of both technologies.

Both families are basically **parsers generators**, because they return models that are able to parse sentences by using annotated data (or **training corpora**) for training. This is why they are data-driven, and they are therefore able to replicate the acknowledged behavior when they have to parse new sentences. Both technologies were established as dominant approaches during the CoNLL Shared Tasks on dependency parsing (Buchholz and Marsi, 2006; Nivre et al., 2007).

### Data Format and Features

The CoNLL data format, which was also established during these Shared Tasks, provides the following attributes that are assumed by the parsers as input and output format:

- The following attributes are used as input:

  1. FORM: Word form. It is basically the word or word form by itself.
  2. LEMMA: Lemma. It is the canonical version of the word, for instance, *speak, speaks, spoke, spoken* and *speaking* come from the same lexeme, with *speak* as the lemma.
  3. CPOSTAG: Coarse-grained part-of-speech tag.
  4. POSTAG: Fine-grained part-of-speech tag. The difference between CPOSTAG and POSTAG is basically that CPOSTAG may be less specific, for instance, for the word *bridge*, the CPOSTAG might be noun, but the POSTAG might be common noun.
  5. FEATS: List of morphosyntactic features (e.g., case, number, tense, etc.). The features are normally separated by a vertical bar or pipe '|'.

  Some input attributes may be empty or not available, the parser should only use the ones that are available in the data set. For instance, some corpora do not have the FEATS or the LEMMA columns available, and others have identic CPOSTAG and POSTAG columns.

- These are the attributes that must be produced by the parsers, except for the last two that are only produced in some cases when needed:

  1. HEAD: Dependency head of the current token. (ID of the HEAD).
  2. DEPREL: Dependency relation to head.
  3. PHEAD: Projective head of the current token. The PHEAD column guarantees a projective dependency structure, which is not the case of the HEAD column. It is not available in most of the corpora having normally an underscore in the last two columns (PHEAD and PDEPREL).

4. PDEPREL: Projective dependency relation to head. It is the projective dependency relation derived from PHEAD if available.

In Figure 1.1 we show an example of a sentence annotated in the CoNLL data format, in which each column shows the information listed above following the same order.

```
1       No      no      r       rn      _       3       NEG     3       NEG
2       habrían haber   v       va      num=p|per=3|mod=i|tmp=c 3               _       3       _
3       pasado  pasar   v       vm      gen=m|num=s|mod=p       0       ROOT    0       ROOT
4       más_de  más_de  r       rg      _       5       _       5       _
5       seis    seis    d       dn      num=p|gen=c     6       _       6       _
6       meses   mes     n       nc      gen=m|num=p     3       SUJ     3       SUJ
7       desde   desde   s       sp      for=s   3       CC      3       CC
8       su      su      d       dp      num=s|per=3|gen=c       9       _       9       _
9       boda    boda    n       nc      num=s|gen=f     7       _       7       _
10      .       .       F       Fp      _       3       PUNC    3       PUNC
```

Figure 1.1: The sentence written in Spanish, "No habrían pasado más de seis meses desde su boda" *[There would have not been more than six months, since his/her wedding]*, annotated in the dependency CoNLL data format.

The training corpus contains all the information, even the output values, for each token,[3] providing the parser with all the information to learn the expected behavior. However, the **test data sets** (the blind ones) must remove the information contained in the last four columns, the parser is responsible of generating them.

The data-driven parsers use **features** in order to define which attributes of the annotated data are useful for parsing constructing a **feature model** which is a subset of the **feature space**, however each parsing family does it in a different way. The features are usually based on the list shown above, even the DEPREL relations, but not the HEAD values. So they are mainly the part-of-speech annotation, the lemma of the words, the original words, morphological information such as gender or number, but they can also be features based on history and the partially built dependency structures.

The attributes LEMMA and FEATS are not available in all data sets, and the CPOSTAG and POSTAG tags are sometimes identical. Note also that the DEPREL attribute is only available dynamically in the partially built dependency tree, this fact make the features based on DEPREL, history-based features. In Chapter 5, we will refer to features that are defined relative to the partial parse as dependency tree features.

The transition-based models use these features to decide which transitions are more likely to occur given a parsing state, however a graph-based model uses mainly the features to make the decisions based on giving weight to every arc contained in the set of possible arcs between the words. Therefore, the goal is the same but it is achieved in a different manner.

[3] In several cases, the columns may be empty, and therefore, not available.

### Evaluation

There are several ways of evaluating the parsers to get the **parsing accuracy**, we usually have a training corpus (for each language) used to let the system learns about the annotation provided. We also have a **test data set** (two versions, one containing all the information, and the other in which the information produced by the parsers is removed) which is rather small and serves as a gold standard to compare between parsers. The following evaluation measures are the ones used and provided in this thesis and in the state-of-the-art papers.

- **LA**: Label Accuracy. It measures the percentage of scoring tokens[4] for which the system has predicted the correct dependency labels.
- **UAS**: Unlabeled Attachment Score. It measures the percentage of scoring tokens for which the system has predicted the correct head.
- **LAS**: Labeled Attachment Score. It measures the percentage of scoring tokens for which the system has predicted the correct head and the correct dependency label.
- **UCM**: Unlabeled Complete–Match. It measures the percentage of sentences for which the system has predicted the correct unlabeled graph.
- **LCM**: Labeled Complete–Match. It measures the percentage of sentences for which the system has predicted the correct labeled graph.

The LAS (or labeled attachment) score is the most common evaluation measure, this is why in most of the cases this is the one provided. However, in other cases, it is worth to find some more information by using something different. There are also other measures that only focus on some parts of the dependency trees, such as **precision** and **recall** for left, right or root attachments. Recently, some researchers are trying to find new ways of evaluting parsers across domains and technologies (Tsarfaty, Nivre, and Andersson, 2011; Tsarfaty, Nivre, and Andersson, 2012).

### Conclusion

Finally, it is worth mentioning that in this thesis, we mainly made use of the transition-based algorithms, basically the ones provided in MaltParser (Nivre, Hall, and Nilsson, 2006),[5] but we also used MSTParser (McDonald et

[4] The "scoring" tokens may vary from one task to another. In the 2006 Shared Task, the punctuation symbols were not included, however in the 2007 Shared Task they were included.

[5] See Section 2.1.1, to find a complete description of MaltParser.

al., 2005; McDonald, Lerman, and Pereira, 2006) and other parsers, always as a matter of comparison. In one of the applications we used Minipar (Lin, 1998), which is a rule-based parser.

## 1.2 Objectives

The present thesis tries, by using state of the art systems (mainly MaltParser) derived from the CoNLL-X (2006) and CoNLL 2007 tasks and other sources, to show some contributions in the following:

First, studying dependency parsing, trying to find whether the performance of the parsers is homogeneus and limitations about corpora size and sentence length.

Second, showing studies and ways of improving the parsing accuracy by modifying the flow of analysis and parsing some segments of the sentences separately. And, more important, by modifying the behavior of state-of-the-art parsers.

Third, investigating automatic feature selection and parsing optimization for transition-based parsers which are considered an important problem and something that definitely needs to be done in current dependency parsing in order to solve all the previous and future problems.

Fourth, dependency parsing are applied to face some interesting NLP problems, such as text simplification and the inference of the scope of negation.

During the realization of this thesis we tried to answer the following questions:

- Which factors in the training corpus can affect the accuracy in the analysis phase?
- Is it possible to improve accuracy by manipulating the training corpus?
- Is it possible to improve accuracy by combining different parsers that are produced by the same parser generator?
- Is it possible to modify the behavior of transition-based parsing algorithms with the intention of improving the performance?
- Is it possible to modify the behavior of graph-based parsing algorithms with the intention of improving the performance?
- Is it possible to automate the optimization of dependency parsers?
- Is it possible to create an automatic (and accurate) feature selection system for a transition-based parser? or even, a machine-learning based parser?

- Can we use dependency structures to solve NLP problems?

We got interesting results by trying to answer all of them. In this thesis we provide answer for these questions, and in Chapter 7 (Conclusions) we summarize our findings.

## 1.3 Structure of The Manuscript

The remaining part of the thesis is structured as follows:

- Chapter 2 (*Related Work and Background*) shows related work to this thesis, focusing on the statistical dependency parsing problem, it also serves as a background chapter. Appendix A is related to Chapter 2.
- Chapter 3 (*Analyzing Dependency Analysis*) shows our initial studies about data-driven dependency parsing, studying mainly the homogeneity of the parsers and corpora. Appendix B is related to Chapter 3.
- Chapter 4 (*Enhancing Dependency Analysis*) shows our prospective works for improving parsing accuracy with parsing combination and, more succesfully, some behavior modifications of transition-based and graph-based parsing algorithms.
- Chapter 5 (*Optimizing Dependency Analysis*) shows our work in parsing optimization. MaltOptimizer, that finds an optimal configuration for a transition-based parsing generator (MaltParser), is presented. Appendices C and D are related to Chapter 5.
- Chapter 6 (*Applying Dependency Analysis*) shows some prospective work in which we emphasize the usefulness of dependency parsing by applying syntactic dependency structures to solve some NLP problems. Appendix E is related to Chapter 6.
- Chapter 7 (*Conclusions and Future Work*) shows the conclusions of the present thesis, a summary of our findings and several suggestions for future work.

All the research done in this thesis, is intrinsically connected. For instance, the parsing combination problem and also the work done by applying dependency parsers is directly related to the research in parsing optimization. This is because they can be applied in sinergy in future work by using the automatic optimization processes by training new models for these problems, in which the optimization steps were handled manually and were one of the hardest problem to tackle. One of the reasons is that we could come up, as we see in Chapter 5, with better feature models in a lot of cases by using our methods because the search in the feature space is normally too extensive to be handled manually.

## 1.4 Chapter Summary

In this chapter we have introduced the basis of dependency linguistics and dependency parsing. We have also established the objectives of the present thesis and the structure of the manuscript. The whole chapter also serves as a motivation chapter.

# Chapter 2

# Related Work and Background

*I was motivated to be different*
*in part because I was different.*
*Donna Brazile*

In this Chapter we show the related work that is relevant for the present thesis. We present it in a way that serves also as a background chapter in order to emphasize the motivation of the thesis.

## 2.1 Background: Dependency Parsing

As we briefly mentioned in the introduction, nowadays there are a wide range of approaches to the Dependency Parsing problem. Rule–based parsers, which are based on a predefined grammar, are significant, having Minipar (Lin, 1998) as a classic reference. These kinds of parsers are very language–dependent, requiring the development of a complete set of rules for every language to be parsed.

Nonetheless, in the last years machine learning–based parsers have become the most popular ones, as shown in (Buchholz and Marsi, 2006) and (Nivre et al., 2007). Machine-learning permits to have a parser for a new language by training a model, avoiding the development of a whole new parser. But, on the other hand, annotated corpora for training are needed. As we introduced in Section 1.1.2, these parsers were established after the celebration of the CoNLL Shared Tasks on Dependency Parsing in two type of approaches, **graph-based** and **transition-based**, and have become the most implemented ones.

We can also find systems that integrate both approaches, providing the

option of selecting between them, such as the ones presented by Bernd Bohnet (2010; 2012). Moreover, as we show in Section 2.3, there are some developments that also combined these two approaches, in which each perspective learns from the another in order to get a higher accuracy (McDonald and Nivre, 2011).

In the following subsections, we describe graph-based and transition-based technologies, based on a parsing example.

### 2.1.1 Transition-Based Parsing

These parsers are called *transition-based*, because they reduce the problem of parsing a sentence to the problem of finding an optimal path through an abstract transition system, or state machine. It may be similar to shift-reduce parsing, but it incorporates a much broader range of transition systems (Nivre, 2008). Transition-based parsers learn models that predict the next state given the current state of the system, including features over previous parsing decisions and the input sentence. At parsing time, the parser starts in an initial state and changes to the new states greedily, by using the predictions of the model learn from annotated corpora, just until the system reaches a final state.

The transition-based parsing strategy results in efficient parsing, with run-times even linear in sentence length, and also provides the use of arbitrary non-local features based on the current state of the tree, since the partially built dependency tree is fixed in any given state. However, as stated in (McDonald and Nivre, 2007), the greedy inference can also lead to the problem of error propagation due to early predictions that may place the parser in incorrect states.

As a summary, here we show the basic assumptions of this methology:

- A transition-based parser processes the sentence from left to right in a deterministic way.
- It (normally) has two data structures: the stack and the buffer.
- In each step, the parsing algorithm has to select one operation: left-arc, right-arc, shift (reduce) or swap. The operations are done from/to the nodes that are in the stack to/from the nodes that are in the buffer.
- A machine learner classifier (often, linear or nonlinear support vector machine) is used to select the operation in every parsing step.
- The features (part-of-speech, morphology, etc) are used by the machine learner to select the operation in the best way possible.

The Figure 2.1 shows a simple parsing example produced by a transition-based parser.

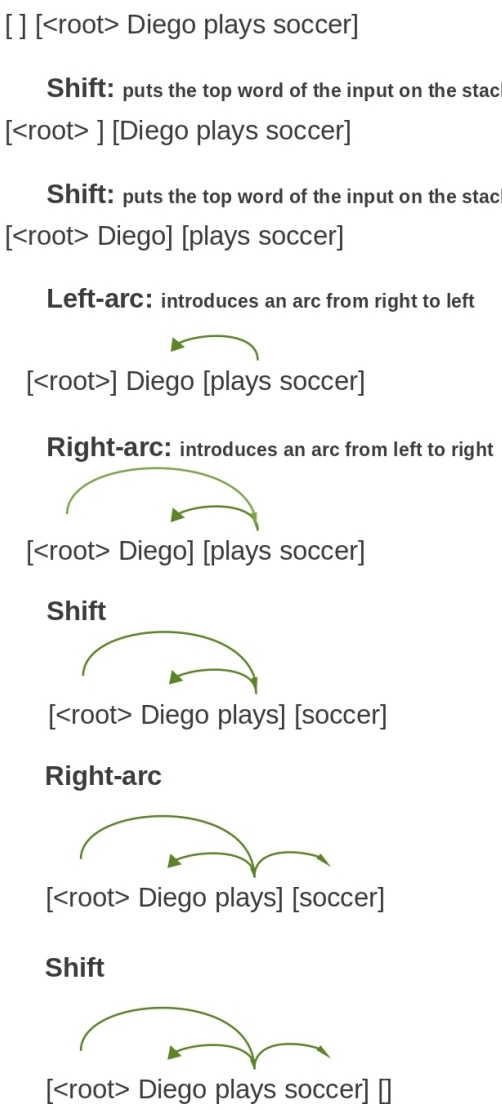



Figure 2.1: Parsing of the sentence *Diego plays soccer* with a transition-based parser. The data structure (between brackets) to the left of the picture is the *stack*, and the one to the right is the *buffer*.

As we said above, one of the main advantages of this kind of parsing perspective is the efficiency, for instance with MaltParser it is possible to perform parsing in linear time for projective dependency trees and in quadratic time in the worst case for non-projective structures (Bosco et al., 2010), however it can even be linear in the average case (Nivre, 2009). The main reason is basically that the search space for each transition is very small, basically the classifier has to select between the possible transitions, normally no

more than 4. Moreover, the lack of backtracking is another reason for the efficiency.

It is worth noting that for the main purposes of this thesis, this is the perspective that is deeply studied (see Chapters 3, 4 and 5). We based most of our studies on MaltParser which is a transition-based parser generator and MSTParser in some of them as a matter of comparison. MaltParser is described with more detail in the following Subsection.

### MaltParser

MaltParser (Nivre et al., 2006a; Nivre et al., 2007) is a data-driven system in which by using a dependency treebank (as the ones presented in Section 2.2), it is possible to train a system capable of parsing a sentence with dependency relations. It implements the transition–based approach to dependency parsing, which, as stated above, is basically a deterministic transition system for mapping sentences to dependency trees and a classifier that predicts the next transition for every possible system configuration.

For training, MaltParser uses support vector machines (Cortes and Vapnik, 1995), either LIBSVM (Chang and Lin, 2001) or LIBLINEAR (Fan et al., 2008). It consists of a single classification decision (create arc, shift, reduce, etc.) for each node.

In MaltParser there are four different algorithm families, all of them transition-based. They provide differences in the way they generate attachments, new data structures and different ways of handling non-projective dependencies. The parsing algorithms available in MaltParser can be grouped into four different families:

1. **Nivre's algorithms**: they include the **arc-eager** and **arc-standard** versions of the algorithm described in (Nivre, 2003) and (Nivre, 2004). The main difference between these two, is the way of handling left attachments, in which the Nivre arc-eager is greedy because it generates left attachments as soon as it can. However, the arc-standard transition system is more lazy when left attachments are about to be generated.

2. **Covington's algorithms**: they include the projective and non-projective versions of the algorithm described by (Covington, 2001) and adapted by (Nivre, 2008). In a way, Covington's algorithms are similar to Nivre arc-eager, concerning parsing order. But they use the data structures in a different way, these structures are even called differently (*left* and *right*, instead of stack and buffer). The non-projective version includes new data structures called the *leftcontext* and the *rightcontext*, containing unattached tokens to the left or right of the first positions of the structures *left* and *right*.

3. **Stack algorithms**: they include the projective and non-projective versions of the algorithm described by Nivre (2009) and by Nivre, Kuhlmann, and Hall (2009). In a way these parsing algorithms are similar to Nivre arc-standard parsing algorithms, concerning parsing order. Nevertheless, in some cases, they include a way of handling non-projective structures in a direct way. The parsing order in this case is lazy, because the parsing transitions are shifted one position and they are therefore postponed to a subsequent state.

4. **Multiplanar parsers** (Gómez-Rodríguez and Nivre, 2010): they include two algorithms; the Planar parser and the 2-Planar parser. These are linear-time algorithms that cover the sets of *multiplanar* dependency structures described by Yli-Jyrä (2003): while the Planar parser is limited to dependency graphs with no crossing arcs between the nodes, which are a rather tight superset of projective dependency graphs, the 2-Planar parser allows the parsing of graphs that can be divided in 2 different planars, allowing non-projective structures.

While the Covington's, the Stack and Multiplanar families contain algorithms that can handle non-projective trees, the Nivre's family does not. However, any projective parsing algorithm can be used to parse non-projective trees by using the technique known as pseudo-projective parsing (Nivre and Nilsson, 2005), obtaining competitive results, similar to the ones obtained by non-projective algorithms. The pseudo-projective parsing technique consists in applying graph transformations before parsing and after transforming the output in post-processing.

MaltParser uses feature models that may be based on the constructed dependency tree and the input string. They are used to predict the next action or transition when the parser reaches a certain parsing state. This fact means that it uses features of the partially built dependency structure along with features of the (tagged) input string. More precisely, features are defined in terms of the wordform, part-of-speech, coarse-grained part-of-speech, dependency relations, list of morphosyntactic features of a token defined relative to one of the data structures. A feature model is defined in an external feature specification file by using a feature specification language[1] and it allows feature models of arbitrary complexity.[2]

### 2.1.2 Graph-Based Parsing

The graph-based parsing models solve the same problem of parsing a sentence but in a rather different way. The main idea is basically that the

[1] An in–depth description of this language can be found in *http://maltparser.org/userguide.html*

[2] In Section 1.1.2 we show a description of the feature models.

parser generates all the possible attachments between lexical items, and afterwards a classifier (often SVM) selects the edges that are annotated with a higher score than the others. In this way, the parser extracts from this super graph, the dependency structure that conforms the dependency tree of the sentence, respecting at the end all the constraints shown in Section 1.1.1.

The Figure 2.2 shows a simple parsing example produced by a graph-based parser.

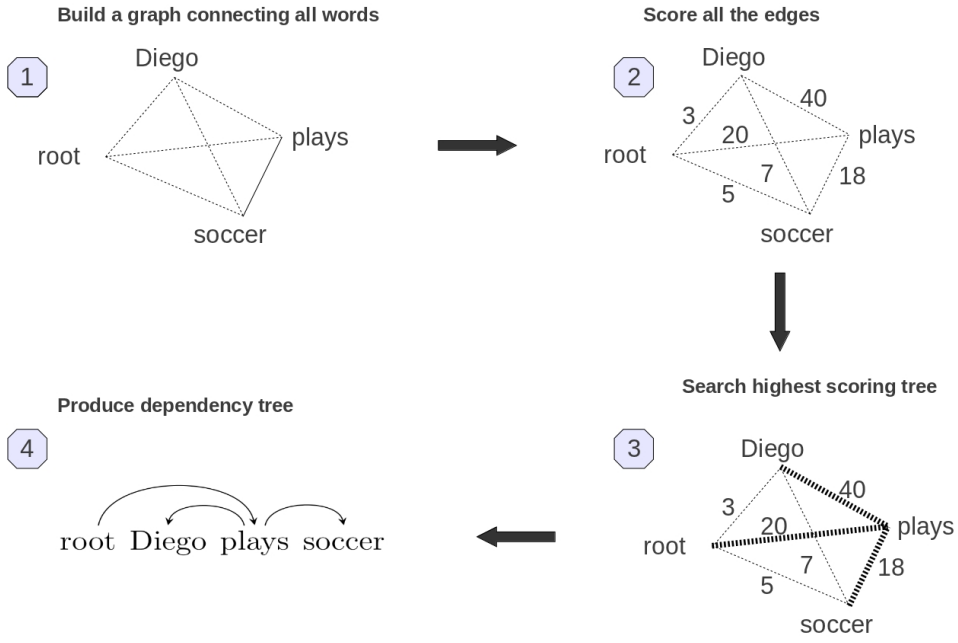



Figure 2.2: Parsing of the sentence *Diego plays soccer* with a graph-based parser.

The graph-based parsing perspective normally requires more time than the transition-based parsing models, more memory and it could even have disk space issues. This is basically the real bottleneck of this perspective, because the transition-based models are much more efficient. A graph-based model requires to store a lot of information, forcing the classifier to select between all the possible edges. In the example shown in Figure 2.2 we only observe 6 possible (and different) arcs.[3] However we can imagine how big the search space can be when we have long sentences with more words, the number of arcs grows substantially. In fact, it is normally cubic or even higher in training time.

[3]Only taking into account undirected arcs, which is not the real case in dependency parsing as we can observe in Section 1.1.

## 2.2 The CoNLL–X and CoNLL 2007 Shared Tasks

The CoNLL 2006 (commonly known as CoNLL-X) and the CoNLL 2007 Shared Tasks were held at the Conference of Natural Language Learning (CoNLL) (Buchholz and Marsi, 2006; Nivre et al., 2007). See Appendix A where we show the results of the CoNLL-X Shared Task in detail. As we stated in Section 1.1, they served to set up a common ground input for the parsers and to establish the state of the art at that time. The CoNLL 2007 also included a domain adaptation task. The domain adaptation task is the one in which the parser should be able to adapt to a new domain by training it over new data.

The two parsers that provided the best results in these Shared Tasks were MSTParser and MaltParser, which are already described above. This is basically the main reason why we used these two parsers in order to run our experiments. We ran most of our experiments with MaltParser and some control experiments with MSTParser. These experiments are shown in the following Chapters.

One of the main contributions of these Shared Tasks is the data format, which is right now an accepted standard for most of the syntactic dependency parsers, and also semantic parsers from the CoNLL 2008 and 2009 Shared Tasks (Surdeanu et al., 2008; Hajič et al., 2009).[4] Very similar formats are even used as a standard for other Shared Tasks, such as the *SEM Shared Task (Morante and Blanco, 2012).

There were 13 annotated corpora in the CoNLL–X Shared Task and 10 annotated corpora in the CoNLL 2007 Shared Task. One for each proposed language. The corpora that have the same language as a source in both Shared Tasks are from the same origin, but they were split in a different way or they were slightly corrected if needed. These treebanks are very important for the present thesis, in which we studied all of them and made use of them from several perspectives in our work. As mentioned in (Buchholz and Marsi, 2006) and in (Nivre et al., 2007) and described in (Abeillé, 2003) the corpora of the CoNLL–X (2006) and CoNLL 2007 Shared Task are the following:

- **Arabic** (2006 and 2007). The Prague Arabic Dependency Treebank (PADT) (Hajič et al., 2004; Smrž, Šnaidauf, and Zemánek, 2002) contains texts in Modern Standard Arabic.
- **Basque** (2007). The Basque dependency treebank (3LB) (Aduriz et al., 2003) contains literary and media texts.
- **Bulgarian** (2006). The Bulgarian dependency treebank (Bultreebank) (Simov, Popova, and Osenova, 2002; Simov et al., 2002; Simov,

[4] See Section 1.1.2 to find the attributes included in the data format or visit *http://nextens.uvt.nl/depparse-wiki/DataFormat*

Osenova, and Slavcheva, 2004; Simov et al., 2005) contains texts in Bulgarian.

- **Catalan** (2007). The Catalan section of the CESS-ECE Syntactically and Semantically Annotated Corpora (Martí et al., 2007), in its origins it was a constituency treebank, but it was automatically converted to dependency annotations by Lluís Màrquez and Antònia Martí.

- **Chinese** (2006 and 2007). The Sinica Treebank (Chen et al., 2003) is a syntactic structure–tagged corpus for Chinese.

- **Czech** (2006 and 2007). The Prague Dependency Treebank (PDT) (Böhmová et al., 2003) consists of a large amount of Czech texts with complex and interlinked morphological structures.

- **Danish** (2006). The Danish Dependency Treebank (Kromann, 2003) is a freely available resource, including parts–of–speech and syntactic annotation.

- **Dutch** (2006). The Alpino Treebank (Van der Beek et al., 2002b; Van der Beek et al., 2002a) is derived from a subset of the Eindhoven Corpus and includes both part–of–speech and dependency annotation.

- **English** (2007). The sections 2-11 and the section 23 of the Penn Treebank (Marcus, Marcinkiewicz, and Santorini, 1993) converted to dependencies by Ryan McDonald. It consists of sentences from the Wall Street journal.

- **German** (2006). The Tiger Treebank (Brants et al., 2002) contains German newspaper texts from the *Frankfurter Rundschau*.

- **Greek** (2007). The Greek Dependency Treebank (Prokopidis et al., 2005) contains Modern Greek sentences.

- **Hungarian** (2007). The Szeged treebank (Csendes et al., 2005) contains fiction texts and newspapers from the legal market.

- **Italian** (2007). The Italian Syntactic-Semantic Treebank (ISST) (Montemagni et al., 2003) contains texts from the *Corriere della Sera* and it uses information from two different domains: general language usage and financial texts.

- **Japanese** (2006). The Japanese Verbmobil Treebank (Kawata and Bartels, 2000) is a syntactically annotated corpus based on spontaneous dialogues, which were manually transliterated.

- **Portuguese** (2006). The Bosque part of the Floresta Sintá(c)tica (Afonso et al., 2002) consists of European and Brazilian Portuguese–language texts automatically annotated by the parser *Palavras* (Bick, 2000) and manually corrected in a postprocessing step.

- **Slovene** (2006). The Slovene Dependency Treebank (SDT) (Džeroski et al., 2006) contains sentences extracted from the translation to Slovene of George Orwell's *1984*.

- **Spanish** (2006). AnCora (3LB) (Civit and Antònín, 2002; Navarro et al., 2003; Civit et al., 2003; Palomar et al., 2004) consists of media texts and literary sentences.

- **Swedish** (2006). Talbanken05 (Nilsson, Hall, and Nivre, 2005) consists of written (proffesional prose and essays) and spoken Swedish (interviews).

- **Turkish** (2006 and 2007). The Metusabanci Treebank (Oflazer et al., 2003; Atalay, Oflazer, and Say, 2003) consists of sentences that were taken from the METU Turkish Corpus.

From each one of these corpora the organizing committee of the CoNLL Shared Tasks extracted a test set, all the test sets being more or less of the same size in wordforms. The remaining part of each source corpus was provided to the participants as the training corpus for each proposed language. The sizes of the training corpora were (and are) not the same, as shown in Table 2.1. In this Table we also show other characteristics of the corpora from the CoNLL Shared Tasks that are interesting for several purposes studied in Chapters 4 and 5: the percentage of non-projective trees and the percentage of left attachments.[5]

## 2.3 Methods on Parsing Combination

In Chapter 4, more specifically in Section 4.1, we show a feasibility study for a parsing combination system. We believe that due to this work it is interesting to show some related and interesting contributions that are also relevant for other studies shown in this thesis.

Despite an active research community devoted to multilingual dependency parsing, there are few studies on improving parsing accuracy without modifying machine learning–based parsers. They can be divided in three different groups. Here we describe the most relevant ones related to this thesis:

[5]This information can also be found in http://ilk.uvt.nl/conll/paper_submission.html and (Buchholz and Marsi, 2006; Nivre et al., 2007).

Table 2.1: Number of sentences, number of wordforms, percentage of non-projective sentences and percentage of left attachments of each training corpus of the CoNLL Shared Tasks

| **Language** | **#Sentences (k)** | **#Wordforms (k)** | **% non-projective sentences** | **% left attachments** |
|---|---|---|---|---|
| Arabic (2006) | 1.4 | 54.3 | 11.2 | 82.9 |
| Arabic (2007) | 2.9 | 112.0 | 10.1 | 79.2 |
| Basque (2007) | 3.2 | 51.0 | 26.2 | 44.5 |
| Bulgarian (2006) | 12.8 | 190.2 | 5.4 | 62.9 |
| Catalan (2007) | 15.0 | 431.0 | 2.9 | 60.0 |
| Chinese (2006) | 57.0 | 337.0 | 0.0 | 24.8 |
| Chinese (2007) | 57.0 | 337.0 | 0.0 | 24.7 |
| Czech (2006) | 72.7 | 1,249.4 | 23.2 | 50.9 |
| Czech (2007) | 25.4 | 432.0 | 23.2 | 46.9 |
| Danish (2006) | 5.1 | 94.3 | 15.6 | 75.0 |
| Dutch (2006) | 13.3 | 195.0 | 36.4 | 46.5 |
| English (2007) | 18.6 | 447.0 | 6.7 | 49.0 |
| German (2006) | 39.2 | 699.6 | 27.8 | 50.9 |
| Greek (2007) | 2.7 | 65.0 | 20.3 | 44.8 |
| Hungarian (2007) | 6.0 | 132.0 | 26,4 | 27,4 |
| Italian (2007) | 3.1 | 71.0 | 7,4 | 65,0 |
| Japanese (2006) | 17.0 | 151.4 | 5.3 | 8.9 |
| Portuguese (2006) | 9.0 | 206.6 | 18.9 | 60.3 |
| Slovene (2006) | 1.5 | 28.7 | 22.2 | 47.2 |
| Spanish (2006) | 3.3 | 89.3 | 1.7 | 60.8 |
| Swedish (2006) | 11.0 | 191.4 | 9.8 | 52.8 |
| Turkish (2006) | 4.9 | 57.5 | 11.6 | 6.2 |
| Turkish (2007) | 5.6 | 65.0 | 33.3 | 3.8 |

1. Parsing combination by stacking: Nivre and McDonald (2011) used stacking to integrate graph–based and transition–based dependency parsers. Stacking is a method in which one or more than one parser learn from one another, by using the output given from a set of parsers and making use of a stack of parsers. The idea is that all the complementary models can learn from one another. This work inspired us to develop the work shown in Section 4.1. Also, we took their study on the lengths of sentences as the starting point for our initial work to study dependency parsing accuracy shown in Chapter 3 (Sections 3.1.1 and 3.2). Moreover, they also studied the progress that has been made through dependency analysis by means of a deep error analysis. Therefore, they combined and compared the two dominant approaches (transition–based models and graph–based models) towards a better accuracy.

2. Parsing combination by voting: a voting system is the combination of several systems in which a facade system lets them vote for the corresponding output, then the system selects the most voted outcome. Zeman and Zabokrtský (2005) proposed an approach that combines

several dependency parsers for Czech. Their goal was to tell, for each word, which parser is the most likely to pick its dependency correctly, by switching the information with a voting system. This work is similar to our n–version dependency parser proposal shown in Section 4.1, but it differs because we use a single parsing generator, and we select the best output for each part of the input sentence.

3. Parsing combination by dual decomposition: Sagae and Lavie (2006), used graph-based parsing to do ensemble parsing and Koo et al. (2010) used dual decomposition to combine the Chu–Liu Edmonds algorithm[6] with a third–order dynamic programming algorithm.

Another relevant work on improving dependency parsing accuracy is (Chen et al., 2009), which shows an approach in which English and Chinese parsing results are enhanced by means of subtrees from self-training data; then new subtree–based features for parsing algorithms is constructed. This work is also related to ours. They studied prepositions as function words and how to coordinate conjuntions, as we show in Section 4.1 but in a different manner. Also, we considered their way of using subtrees and their idea of trying to get a higher complete–match accuracy, as we study in Section 3.4.

## 2.4 Parsing Optimization and Machine-Learning

In Chapter 5, we show a system that optimizes parsing models automatically, there is some related work about machine learning and automatic feature selection which is directly related with our work. This work is also related with manual feature selection and the feature models that we used in Chapter 3 and Chapter 4.

Automatic feature selection for transition-based dependency parsing was recently explored in a study by (Nilsson and Nugues, 2010). Starting from a much more reduced feature model than the MaltParser default models, they add features incrementally using a notion of topological neighbors in the feature space. They also experiment with different levels of greediness and report competitive results on three different data sets.

In natural language processing more generally, optimization problems have been studied by (Kool, Zavrel, and Daelemans, 2000) and (Daelemans et al., 2003), in which they use genetic algorithms for model selection in the context of part-of-speech tagging, grapheme-to-phoneme conversion with stress assignment, and word sense disambiguation. They studied feature selection together with parameter optimization, trying to reach joint optima. A tool developed specifically for the optimization of learning algorithm parameters in the context of natural language processing is Paramsearch (van den Bosch, 2004).

[6] http://www.softpanorama.org/Algorithms/Digraphs/mst.shtml

In machine learning more generally, the feature selection problem has been the object of numerous studies (Guyon and Elisseeff, 2003; McCallum, 2003), and greedy methods like the ones employed by ourselves in Chapter 5 are well represented in the literature. For instance, (Korycinski et al., 2003) use an adaptive greedy feature selection technique to solve the hyperspectral data analysis in the optical engineering area, or (Doraisamy et al., 2008) that present a comparative study on feature selection techniques applied to automatic genre classification, including best first, greedy stepwise and genetic adaptative search, and conclude that all these methods can significantly improve the performance of general machine learning models. Moreover, (Pahikkala, Airola, and Salakoski, 2010) show how to speed up forward feature selection by applying greedy search, using a strategy similar to that employed by us in Chapter 5. Finally, (Das and Kempe, 2011) demonstrate that greedy algorithms perform well even when the features are highly correlated, which is something that definitely happens in transition-based dependency parsing.

## 2.5 Chapter Summary

We show here a brief summary of relevant research related to the work presented in the thesis.

This thesis is mainly based in transition-based dependency parsing, more concretely, it is inherently related with **MaltParser** (Nivre, Hall, and Nilsson, 2006). This parser generator is constantly improved,[7] and it is an open-source system for data-driven, and transition-based, dependency parsing that offers a wide range of parameters for optimization. First of all, it implements nine different transition-based parsing algorithms, each of which has its own parameters. Secondly, for each parsing algorithm it is possible to define arbitrarily complex feature models using an expressive feature specification language. Finally, any combination of parsing algorithm and feature model can be combined with a number of different machine learning algorithms available in LIBSVM (Chang and Lin, 2001) and, more recently, with LIBLINEAR (Fan et al., 2008). See Section 2.1.1 in order to find a more complete description.

We have also investigated and made use of other parsers. A clear example is **MSTParser** (McDonald et al., 2005; McDonald, Lerman, and Pereira, 2006), we mainly used this parser generator as a matter of comparison with the results that we produced in some of the cases with MaltParser, when it was needed. MSTParser belongs to the graph-based parsing perspective. See Section 2.2 in order to find a more complete description.

In the field of parsing combination there are several contributions that inspired our work, such as the one done by Joakim Nivre and Ryan McDon-

[7] At this writing, the last version (1.7.1) is dated on April 2012

ald integrating graph–based and transition–based dependency parsers (2008; 2011), comparing and combining the results of Maltparser and MSTParser. They showed how feasible it is to integrate both systems by using stacking at training time instead of a voting system. Also, we took their study on the lengths of sentences as a starting point for our first work studying dependency parsing, which is shown in Chapter 3 (Sections 3.1.1 and 3.2).

In the context of parsing optimization it is hard to find publications that try to face this problem in the NLP community. We certainly believe that this will change in the near future because data-driven systems for natural language processing have the advantage that they can easily be ported to any language or domain for which appropriate training data can be found, but, such systems require careful tuning in order to achieve an optimal performance, which may require specialized knowledge of the system and the task that is being solved in each case. This is an unsolved issue in current dependency parsing and we therefore believe that we will see more research about this in the future. In transition based dependency parsing we can find an example of automatic feature selection carried out by making use of a forward selection of features (Nilsson and Nugues, 2010), in which they come up with an optimal feature set for a single arc-eager MaltParser algorithm. There are optimization tools for general machine learning algorithms, such as Paramsearch (van den Bosch, 2004).

In this Chapter we have shown the relevant work to this thesis, in a way in which the reader can go backwards and find here the citations and some important concepts that might be needed to read the present thesis, serving also as background chapter. It is worth noting that some related work is relevant only for specific sections later in the thesis, and it is described in the corresponding chapters later on.

# Chapter 3

# Analyzing Dependency Analysis

*The best preparation for good work*
*tomorrow is to do good work today.*
*Elbert Hubbard*

In this chapter we show some technical analyses in which we studied dependency parsing, based on MaltParser, and its limitations:

1. We present our study about the homogeneity of the Spanish dependency parsing problem focusing on training corpora size and sentence length. In Sections 3.1 and 3.2.

2. We present our study about the limitations of the CoNLL-X Shared Task corpora. Section 3.3.

3. We present our study reevaluating several parsers by using complete-match evaluation measures, because we consider that they should be taken into account in future parsing developments. In Section 3.4.

Most of the experiments shown in this Chapter, in particular in Sections 3.1, 3.2 and 3.3 were conducted with MaltParser in default settings and the specific feature models published for the CoNLL-X Shared Task.[1] We manually translated the old feature models to the new feature model language (included in the latest MaltParser versions), and we always used the current version of MaltParser which is updated (about) every three months.

For the experiments in which we show the output of different parsers, as the one shown in Section 3.4, we used the published outputs of the CoNLL-X

[1] In the website of MaltParser (http://www.maltparser.org/conll.html) it is possible to find a set of feature models for the treebanks of the CoNLL Shared Tasks

Shared Task in order to carry out a fair comparison.[2]

## 3.1 Study of the Spanish Accuracy: The Homogeneity Assumption

Our first goal was to study accuracy by considering corpus factors. Inspired by (Herrera and Gervás, 2008) we considered that training corpus size is an area that must be considered while studying dependency parsing accuracy. After replicating the results obtained by Nivre et al. (2006a) in the CoNLL–X Shared Task, we formulated the following hypothesis: it is possible to enhance the accuracy by manipulating the training corpus.

In this Section we show our studies about the homogeneity of the accuracy, by modifying the training corpus and by trying to show that the selection of the best subset for training is something that is worth studying.

### 3.1.1 First Experiment: How Homogeneous is the Accuracy?

Regarding a way to study training corpora, the following question arises: *Can we expect the same results in terms of accuracy for every parsed text with a model trained with MaltParser?* In order to find an answer to this question we conducted the experiment shown below.

#### Configuration of the First Experiment

First of all, we divided the whole Spanish corpus into 21 disjoint subsets of about 4,500 wordforms each. These subsets have the following characteristics:

- A similar size, in terms of wordforms, to the test set used in the CoNLL–X Shared Task (4,500 wordforms). This division was done for reasons of comparison, because each subcorpus was used in the experiment as a test corpus. In this way we had test corpora in similar conditions to the ones provided in the CoNLL–X Shared Task.

- Each subcorpus contained a similar number of sentences as the ones present in the original Spanish test data set of the CoNLL-X Shared Task. This fact means, that the sentences of the Spanish corpus are distributed homogeneously among the 21 subsets according to their length. We therefore sampled pseudo randomly, which provides much more homogeneous divisions.

[2] See http://ilk.uvt.nl/conll/online_results.tar.bz2

We trained MaltParser with each subcorpus, so we got 21 models ready to parse. With each model we parsed the other 20 subsets that were not used to train the corresponding model. In this way we obtained $21 \times 20 = 420$ parsed corpora. After that, we evaluated the resulting outcomes. As evaluation metrics we computed not only LAS, UAS and LA, we also used the following set of metrics that we considered useful to extract further conclusions.

- The correlation coefficients between:
  - The LAS data series and the UAS data series performed by each model ($r_{LAS,UAS}$).
  - The LAS data series and the LA data series performed by each model ($r_{LAS,LA}$).
  - The correlation coefficient between the UAS data series and the LA data series performed by each model ($r_{UAS,LA}$).

  By computing these coefficients we wanted to investigate whether, in spite of the differences between the parsed subsets of text, a certain correlation existed between each pair of metrics.

- The maximum and the minimum values for each case:
  - LAS ($max_{LAS}$, $min_{LAS}$).
  - UAS ($max_{UAS}$, $min_{UAS}$).
  - LA ($max_{LA}$, $min_{LA}$).

With this set of metrics we wanted to study the stability of the accuracy achieved by MaltParser when it is trained with a corpus from the CoNLL-X Shared Task, in this case, the Spanish one.

### Results of the First Experiment

Table 3.1 shows the evaluation results of the 20 corpora evaluated by considering correlation coefficients, maximum values and minimum values.

If we analyze the second column of Table 3.1 ($r_{LAS,UAS}$) we can conclude that in most cases the correlation between LAS and UAS is high. The same conclusion between LAS and LA can be obtained by observing the values in the third column ($r_{LAS,LA}$). These values are logical if we consider the definitions of LAS, UAS and LA, i.e., LAS results depend simultaneously on a correct attachment and a correct labeling. The correct attachment is directly related to UAS and the correct labeling is directly related to LA. But when analyzing the fourth column ($r_{UAS,LA}$) we observe very different values, from 0.44 to 0.91; this can be also explained by considering the definitions for UAS and LA, i.e., UAS does not depend on the labeling, so

| **Sub–corpus** | $\mathbf{r}_{LAS,UAS}$ | $\mathbf{r}_{LAS,LA}$ | $\mathbf{r}_{UAS,LA}$ | $\mathbf{max}_{LAS}$ | $\mathbf{min}_{LAS}$ | $\mathbf{max}_{UAS}$ | $\mathbf{min}_{UAS}$ | $\mathbf{max}_{LA}$ | $\mathbf{min}_{LA}$ |
|---|---|---|---|---|---|---|---|---|---|
| $A_0$ | 0.84 | 0.78 | 0.44 | 72.78% | 69.06% | 78.22% | 74.20% | 85.03% | 82.45% |
| $A_1$ | 0.87 | 0.85 | 0.65 | 72.86% | 68.81% | 77.29% | 74.88% | 84.85% | 82.36% |
| $A_2$ | 0.92 | 0.87 | 0.91 | 73.55% | 68.40% | 78.47% | 69.04% | 85.85% | 75.24% |
| $A_3$ | 0.92 | 0.80 | 0.54 | 72.68% | 69.01% | 77.58% | 74.42% | 85.39% | 82.35% |
| $A_4$ | 0.88 | 0.87 | 0.71 | 72.02% | 68.96% | 77.09% | 74.31% | 84.71% | 82.45% |
| $A_5$ | 0.82 | 0.89 | 0.61 | 72.74% | 69.32% | 77.32% | 74.90% | 84.99% | 82.40% |
| $A_6$ | 0.90 | 0.88 | 0.69 | 71.90% | 68.42% | 76.88% | 74.32% | 84.78% | 82.20% |
| $A_7$ | 0.86 | 0.88 | 0.63 | 72.55% | 68.16% | 77.27% | 73.61% | 85.24% | 81.95% |
| $A_8$ | 0.94 | 0.87 | 0.71 | 72.92% | 67.71% | 77.55% | 73.65% | 85.52% | 82.03% |
| $A_9$ | 0.91 | 0.76 | 0.56 | 72.27% | 68.23% | 77.47% | 73.99% | 84.85% | 82.35% |
| $A_{10}$ | 0.87 | 0.89 | 0.69 | 71.76% | 68.19% | 77.11% | 73.00% | 84.62% | 81.74% |
| $A_{11}$ | 0.91 | 0.85 | 0.69 | 73.30% | 68.37% | 78.27% | 73.68% | 85.73% | 82.52% |
| $A_{12}$ | 0.92 | 0.78 | 0.60 | 73.01% | 69.27% | 78.32% | 74.62% | 85.39% | 82.43% |
| $A_{13}$ | 0.89 | 0.85 | 0.64 | 72.96% | 69.61% | 78.22% | 74.46% | 85.60% | 82.19% |
| $A_{14}$ | 0.92 | 0.81 | 0.62 | 73.04% | 68.29% | 77.93% | 74.07% | 85.04% | 82.27% |
| $A_{15}$ | 0.94 | 0.85 | 0.76 | 71.37% | 67.81% | 76.21% | 72.60% | 85.01% | 82.42% |
| $A_{16}$ | 0.87 | 0.76 | 0.44 | 72.51% | 68.83% | 76.83% | 73.89% | 85.50% | 82.17% |
| $A_{17}$ | 0.92 | 0.79 | 0.58 | 73.23% | 68.82% | 77.58% | 74.17% | 85.78% | 82.77% |
| $A_{18}$ | 0.95 | 0.78 | 0.63 | 72.50% | 67.40% | 77.63% | 73.18% | 84.70% | 81.82% |
| $A_{19}$ | 0.82 | 0.86 | 0.54 | 72.25% | 68.99% | 77.65% | 74.19% | 85.39% | 82.99% |
| $A_{20}$ | 0.95 | 0.84 | 0.73 | 72.73% | 68.28% | 77.55% | 73.68% | 85.19% | 82.07% |
| **max** | 0,95 | 0,89 | 0,91 | **73,55%** | **69,61%** | **78,47%** | **74,90%** | **85,85%** | **82,99%** |
| **avg** | 0,90 | 0,83 | 0,64 | **72,62%** | **68,57%** | **77,54%** | **73,76%** | **85,20%** | **81,96%** |
| **min** | 0,82 | 0,76 | 0,44 | **71,37%** | **67,40%** | **76,21%** | **69,04%** | **84,62%** | **75,24%** |

Table 3.1: Results obtained by the models trained with the 21 subsets in which Spanish corpus was splitted.

we can incorrectly label a correct attachment and vice versa. If the parser is sufficiently good it should simultaneously accomplish the labeling and the attachment correctly.

If we consider the 21 models, the one showing the maximum variation among its LAS series values is $A_8$ with a difference of 5.21 points. The one showing the minimum variation among its LAS series values is $A_4$ with a difference of 3.06 points. For UAS, the maximum variation is performed by $A_2$ with a difference of 9.43 points, and $A_1$ performs the minimum variation showing a difference of 2.41 points. Finally, $A_2$ reaches the maximum difference not only for UAS but also for LA (10.61 points), while the minimum variation occurs again with the model $A_4$ (with 2.26 points) that obtained the minimum variation for LAS. From these values we can conclude that each model can achieve a relatively wide range of accuracy values depending on the texts that are used as input. But the overall results across the 21 models are more homogeneous, as can be seen in the last three rows of Table 3.1, from the fifth column to the last.

### Conclusions of the First Experiment

After analyzing the results presented it can be concluded that the accuracy is homogeneous. Thus, we can conclude:

- When training two models with different corpora that have a similar size and a similar number of sentences for each sentence length, they should yield similar overall accuracies. However, each model could probably get notably different accuracy values depending on the specific subset of text used as input. This is because the training corpus size and the lengths of its sentences could contribute to the parsing accuracy registered. That led us to carry out the experiments described in Sections 3.1.2 and 3.1.3.

- Better parsing accuracies could be achieved by combining specific parsers. Each specific parser would parse only the subsets of text for which it yields the best accuracy. This idea motivated the experiment shown in Section 3.1.2 and the N-Version parser shown in Section 4.1.

### 3.1.2 Second Experiment: Is a Better Performance Possible?

The accuracies obtained when parsing a subcorpus with different models inspired another study within this experimental set up. This second experiment was configured to verify whether some models induce noise in the precision, meaning that they should be removed from the training corpora, and to find out an answer to the following question: *Are there models that produce better accuracy than others?* If the answer is yes we can conclude that the training corpora must be tagged and selected very carefully to achieve the best accuracy possible with the technology given.

#### Configuration of the Second Experiment

This second experiment was developed using the metrics computed from the 420 models of the previous experiment. If every $A_i$ is parsed with the model trained with $A_j, j \neq i$, that performs the best possible LAS, then we will get the best overall LAS for the whole corpus, i.e., $\bigcup_{0 \leq i \leq 20} A_i$. In short, the best overall LAS can be achieved by combining the actions of all the models in a way that each model will parse only the subsets of text for which they are the best. Table 3.2 shows which model must be used to parse every $A_i$.

Note that in this experiment we trained with small corpora of about 4,500 wordforms each. However, in the previous experiment we trained with a training corpus containing around 90,000 wordforms.

#### Results of the Second Experiment

As it is shown in Table 3.2 some models, trained with some specific subsets, are capable of parsing better than other, different subsets. For instance, the model trained with $A_{17}$ parses, at best, four of the 21 subsets. But there are other models that always return worse results. It could be because

those models contain very different syntactic structures, or simply that these models are parsed in a different way.

| Input | Parsed by model | LAS |
|---|---|---|
| $A_0$ | $A_{14}$ | 72.58 % |
| $A_1$ | $A_{18}$ | 72.20 % |
| $A_2$ | $A_{17}$ | 70.93 % |
| $A_3$ | $A_1$ | 70.99 % |
| $A_4$ | $A_{12}$ | 72.47 % |
| $A_5$ | $A_{13}$ | 70.17 % |
| $A_6$ | $A_2$ | 73.55 % |
| $A_7$ | $A_0$ | 71.58 % |
| $A_8$ | $A_4$ | 71.57 % |
| $A_9$ | $A_{17}$ | 73.23 % |
| $A_{10}$ | $A_3$ | 71.91 % |
| $A_{11}$ | $A_3$ | 71.78 % |
| $A_{12}$ | $A_3$ | 71.27 % |
| $A_{13}$ | $A_7$ | 69.83 % |
| $A_{14}$ | $A_6$ | 71.90 % |
| $A_{15}$ | $A_5$ | 70.24 % |
| $A_{16}$ | $A_5$ | 70.07 % |
| $A_{17}$ | $A_1$ | 70.77 % |
| $A_{18}$ | $A_{12}$ | 71.94 % |
| $A_{19}$ | $A_{17}$ | 71.40 % |
| $A_{20}$ | $A_{17}$ | 70.79 % |
| **Avg LAS** | | 71.48 % |

Table 3.2: Models trained with $A_j, j \neq i$ that must be used to parse each $A_i$ to obtain the best possible overall average LAS.

### Conclusions of the Experiment

The results discussed here may encourage a more complex evaluation for dependency parsing systems, which not only should assess how high the given accuracy is but also the persistence of accuracy values across a wide range of input sets.

As we see that some models are better than others and parse some specific subsets better, this experiment should encourage the development of n–version parsers; see Section 4.1. These parsers should consist of several specific models, each one trained to obtain high accuracy for a small range of sentences. Therefore, the system should select the specific model that would best parse the sentence that is used as input. Of course, it is easy to know *a posteriori* which one is the best parser for a specific subset of text, but having an unknown corpus to parse how do we know *a priori* which one is the best parser for it? Given a whole corpus it is very difficult to provide an answer to this question. But this idea can be implemented at each stage, so we can consider it at the sentence level.

Moreover, as a conclusion for this experiment we can suggest the development of more efficient processes for building training corpora for dependency parsing. This way, unnecessary effort for labeling a number of

sentences could be avoided by carefully selecting the most convenient ones.

### 3.1.3 Third Experiment: Does the Training Corpus Size Affect Parsing Accuracy?

The present subsection shows an experiment focused on the analysis of the effect of the training corpus size on parsing accuracy.

#### Configuration of the Third Experiment

To analyze the effect of training corpus size on parsing accuracy we incrementally built a training corpus and evaluated parsing performance for each model trained, as follows:

- First of all we selected the $A_i$ $(1 \leq i \leq 20)$ for which we obtained the best LAS when we parsed all the $A_i$ $(1 \leq i \leq 20)$ with the model trained with $A_0$ (see the experiment described in section 3.1.2). This subcorpus was $A_6$ and it was the first one added to the incremental training corpus, which initially was empty.
- In every iteration we trained MaltParser with the incremental corpus and we tested the trained model by parsing $A_0$ with it.
- In each iteration we added the unused subcorpus $A_i$ $(1 \leq i \leq 20)$ for which we obtained the best LAS when we parsed it with the model trained with $A_0$ in the experiment described in section 3.1.2.
- We iterated 20 times until each $A_i$ was added to the incremental training corpus. Since each $A_i$ $(1 \leq i \leq 20)$ shows a distribution of sentence lengths proportional to the distribution present in the Spanish corpus, in each iteration the incremental training corpus had a similar average sentence length as any other corpus from a different iteration.

#### Results of the Third Experiment

The results of this experiment are shown in Figure 3.1. Considering LAS, from the first to the second iteration it becomes almost 3 points higher. From the second to the third iteration LAS increases 1.38 points. In the fourth iteration LAS is 1.2 points higher. And it increases almost 1 point after the fifth and the sixth iterations. By adding about 22,600 wordforms to the training corpus that we had in the first iteration we obtained a LAS increment of 7.56 points. But by adding another 22,600 wordforms LAS increments only 1.63 points. Taking into account the variations of LAS showed in Subsection 3.1.2, this last increment of 22,600 words is not very high. So it might seem that after a certain limit the training corpus size does not provide a significant contribution to parsing accuracy.

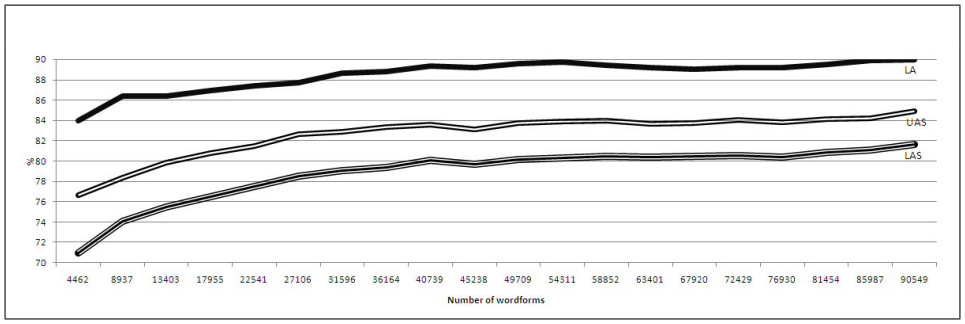



Figure 3.1: LAS, UAS and LA depending on the number of wordforms contained in the training corpus.

#### Conclusions of the Third Experiment

After systematic study of the accuracy increment given when incrementing the size of the training corpus while maintaining the distribution of sentence length, we can conclude that over a certain threshold the amount of words in the training corpus does not meaningfully affect the accuracy achieved. In other words, a training corpus containing all kinds of sentence lengths does not significantly contribute to parsing accuracy after a given size. So it seems that for Spanish parsing, words are not as important as sentences for training corpora. This fact means that when building training corpora, at least for Spanish, it seems more important to include all kinds of sentence lengths, having different syntactic structures, in it, rather than giving priority to its total size. These thoughts could lead us to develop further research on sentence length and optimizing training corpora, as it is shown in Sections 3.2 and 3.3.

## 3.2 The Role of Sentence Length

As shown in (McDonald and Nivre, 2011), parsing systems tend to have lower accuracies for longer sentences, due to the complex syntactic structures that are involved. In this section we want to show that longer sentences are more useful if we consider including them in the training corpora, in order to achieve higher accuracy.

After the previous experiment (in which sentence length was revealed to be a key feature in training corpora for Spanish, because it seems that the system does not learn more when all possible syntactic structures are already included) and considering previous research done by McDonald and Nivre (2011) in which parsing errors related to sentence length are studied we decided to carry out a new experiment focusing on it. Therefore, we developed an experiment to determine whether the length of the sentences contained in the training corpus could affect parsing accuracy. Actually, this

experiment consists of two experiments:

- The first one, described in Subsection 3.2.1, shows a study on the effect of sentence length on accuracy.
- The second one, described in Subsection 3.2.2, compares the accuracy of corpora built exclusively with long sentences and corpora exclusively built with short sentences.

### 3.2.1 Training Corpora Containing Sentences of a Unique Length

This experiment shows a study of the effect of sentence length on accuracy, and it is an attempt to show how sentence length in the training corpus affects parsing accuracy with the following hypothesis: *Longer sentences are more useful for training than shorter ones.*

Figure 3.2 shows the distribution of sentences in the Spanish corpus according to their length; on the x axis we represent the value of the length and on the y axis the number of sentences. It is worth noting that long sentences are very rare in the Spanish treebank. Therefore, in order to find whether the inclusion of all kinds of sentence length in the training corpus is actually a key feature, another question arises: *can we enhance accuracy by building a corpus with sentences of the same length?*

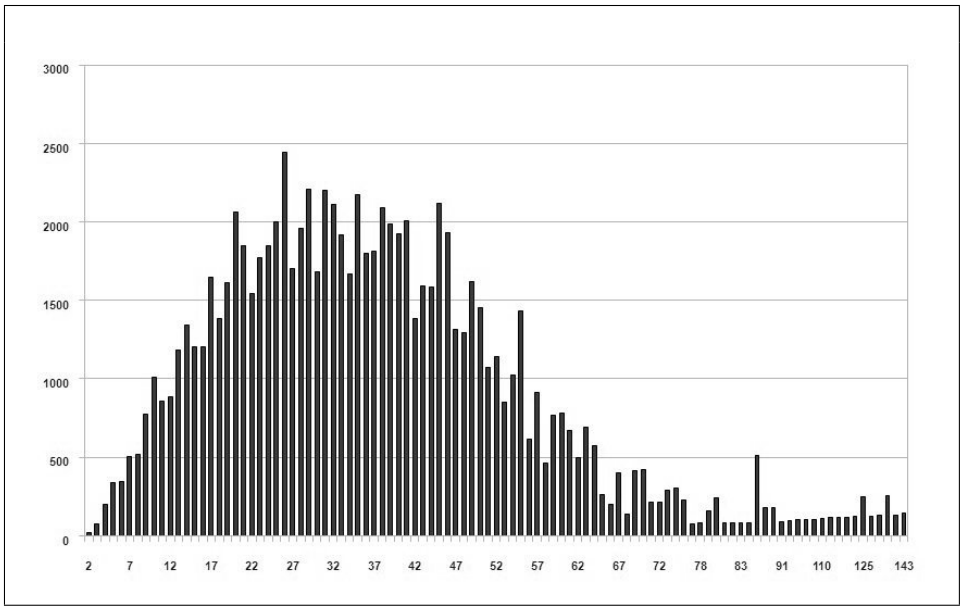



Figure 3.2: Distribution of sentences in the Spanish corpus according to their length.

### Configuration of the experiment

The section of the Spanish corpus that was provided as training corpus in the CoNLL–X Shared Task was divided into 102 subsets, each one containing sentences of a unique length. Thus we obtained a subcorpus with 1 sentence of 143 wordforms, another subcorpus with 1 sentence of 130 wordforms, another subcorpus with 2 sentences of 128 wordforms and so on.

By training MaltParser with each one of these subsets, we obtained 102 different models. With each of them, we parsed the section of the Spanish corpus that was provided as test corpus in the CoNLL–X Shared Task.

### Results of the Experiment

The results of the experiment are shown in Figure 3.3. In this figure we plot LAS, UAS and LA for each parsing. On the x axis we represent the length of the sentences contained in the subcorpus used as a training corpus. LA values are represented by the upper line, UAS values are represented by the middle line and LAS values are represented by the lowest line.

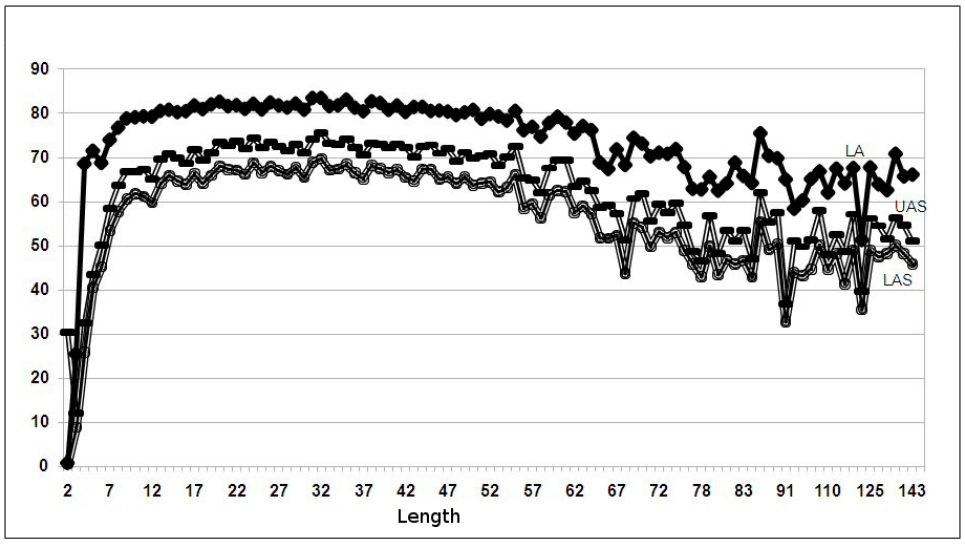



Figure 3.3: LAS, UAS and LA when training with corpora containing sentences of a unique length.

As can be observed, training corpora containing longer sentences gave remarkable accuracy despite its small size (in wordforms). For example a 143-wordform corpus, with only one long sentence, gave us 45.88% LAS, 51.16 % UAS and 66.15% LA tested over the whole test data set.

Taking the results shown above into account and the fact that the Spanish treebank has 35 sentences with 80 or more wordforms,[3] we carried out

[3]Figure 3.2 shows that longer sentences are a very small part of the corpora.

another experiment trying to give an answer to the following question: *Could we get good results while considering only long sentences?* We therefore systematically added sentences, creating new training subsets with only long sentences, and we trained a model with each subcorpus. We started with sentences of 120 wordforms or more, then we repeated the same experiment after adding sentences of 110 wordforms or more to the previous subcorpus. Next we added sentences of 100 wordforms or more; next of 90 wordforms or more and finally of 80 wordforms or more. Table 3.3 shows LAS, UAS and LA values for these subsets. The results are notably high for LAS and UAS, and comparable to the ones obtained for LA, if we compare to the ones obtained with a larger corpus of more than 4,500 wordforms that contained sentences of all kinds of sizes, as we saw in Section 3.1.1.

In order to carry out a control experiment and having a comparison, we repeated the same experiment but only taking into account short sentences. We obtained the results shown in Table 3.4. There are 571 sentences in the Spanish corpus with 10 wordforms or less. As we can observe in the results shown in Table 3.3 and Table 3.4 we can observe how the corpora containing longer sentences are more accurate both in the attachment scores and the labelling scores, we should look into the final size of each generated corpora in order to carry out a fair comparison.

| **Subcorpus** | **Size** | **LAS** | **UAS** | **LA** |
|---|---|---|---|---|
| 120..143 | 1154 | 61.07% | 67.60% | 77.18% |
| 110..143 | 1612 | 64.64% | 70.23% | 80.24% |
| 100..143 | 1919 | 66.68% | 71.99% | 81.41% |
| 90..143 | 2104 | 66.86% | 72.17% | 81.90% |
| 80..143 | 3538 | 68.60% | 74.11% | 82.72% |

Table 3.3: Results obtained by the models trained with the subsets, only longer sentences (size in wordforms).

| **Subcorpus** | **Size** | **LAS** | **UAS** | **LA** |
|---|---|---|---|---|
| 2..7 | 1471 | 57.60% | 62.19% | 78.10% |
| 2..8 | 1991 | 62.16% | 66.86% | 79.75% |
| 2..9 | 2765 | 64.51% | 68.90% | 81.52% |
| 2..10 | 3775 | 66.84% | 72.30% | 82.79% |

Table 3.4: Results obtained by the models trained with the subsets, only shorter sentences (size in wordforms).

It is worth noting that in every case the differences for LAS and UAS are very remarkable, however the results for LA are more or less in the same

range. This result basically means that the part that corresponds to the syntactic structure (evidenced in LAS and UAS) is significantly better with long sentences, because they are rich of different syntactic structures. However, the part that concerns the labelling (evidenced in LA) is not that significantly better, and a corpus with short sentences seems to be as accurate as the ones obtained over corpora containing only long sentences.

#### Conclusions of the Experiment

From the results shown above, it can be concluded that longer sentences, when included in the training corpus, contribute more than shorter sentences to overall accuracy. Quite substantially in the case of the attachments but with a non-significant lack in label accuracy. Thus, a training corpus containing only long sentences needs fewer wordforms than a training corpus containing only short sentences to achieve similar accuracies. We believe that this fact is a very important conclusion and could lead to build corpora in a better way focusing on the quality of the sentences and the annotation and not in the size of the final treebank.

### 3.2.2 Training Corpora Containing the Best Performing Sentences

Next, we developed another experiment focused as well on sentence length. Our goal was to show that longer sentences included in the training corpus are useful to train more accurate models than shorter sentences from a different perspective. This experiments serves to compare whether the accuracy grows faster when we give priority to long sentences in the iterations. This is why, the experiment is set up in a similar way as the experiment shown in Section 3.1.3.

#### Configuration of the Experiment

We incrementally built a training corpus and evaluated the parsing performance for each model trained, as follows:

- First of all, we selected the subcorpus for which we obtained the best LAS value in the previous experiment described in Section 3.2.1.
- In each iteration we trained MaltParser with the incremental corpus (as in 3.1.3) and tested the resulting trained model by parsing the section of the Spanish corpus that was provided as test corpus in the CoNLL–X Shared Task with it.
- In each iteration we added to the incremental corpus the non–used subcorpus for which we obtained the best LAS value in the previous experiment described in this section.

- We iterated 102 times until every subcorpus was added to the incremental training corpus.

### Results of the Experiment

In Figure 3.4 we show LAS, UAS and LA values for each parsing. On the x axis we represent the number of wordforms contained in the incremental training corpus in each iteration. In the Figure we can observe how with half of the training set we are approaching the maximum accuracy that we can get at the end, and it reaches the maximum much faster than the experiment shown in Section 3.1.3, summarized in Figure 3.1.

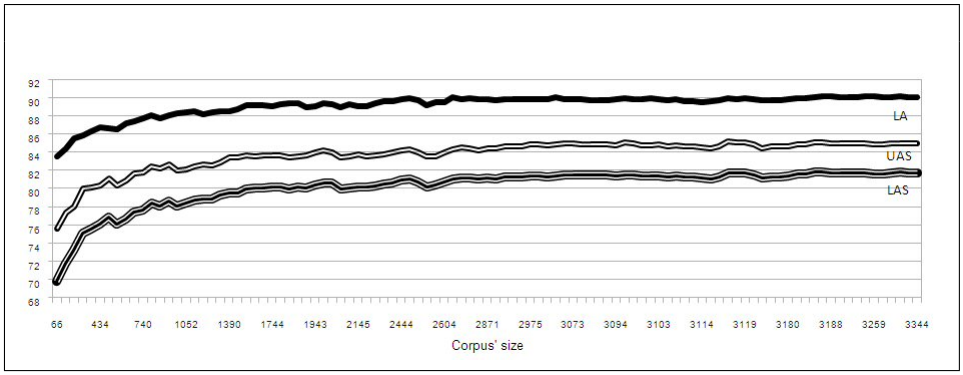



Figure 3.4: LAS, UAS and LA when considering sentence length to build the training corpus.

### Conclusions of the Experiment

From the results shown above, we can conclude that the selection of training corpus sentences according to the length permits the acquisition of better overall LAS, UAS and LA with a smaller number of wordforms than when sentence length is not considered. We can observe that the accuracy is established with the half of sentences that are contained in the whole corpus. It means that models generated with corpora containing short sentences are not as useful for training as models generated with longer sentences. If we compare the results of this experiment, shown in Figure 3.4 and the results of the experiment 3.1.3, shown in Figure 3.1, accuracy grew much faster in this second experiment than in the first. This is because in this second experiment, with a similar set-up, we first selected the longer sentences, because they were the ones that produced better results in the previous experiment.

## 3.3 Study of Training Corpora Size for Multiple Languages

This Section shows another experiment that addresses the problem of the size of the training treebank data for a big set of languages (the ones from the CoNLL-X Shared Task). We believe that training corpora size is always the bottleneck for the purposes of training, considering time and memory constraints. Therefore, this experiment was motivated by the ones shown in Sections 3.1 and 3.2, in which we addressed the same problem from a different perspective and a single corpus. In the previous experiments we realized that it seems that current corpora used for training machine learning–based dependency parsers contain a significant proportion of information that is not needed in order to get a high accuracy.

Since the development of such training corpora involves a large effort, we argue that an appropriate process for selecting the sentences to be included in them can result in having parsing models as accurate as the ones given when training with bigger – non optimized corpora (or alternatively, higher accuracy for an equivalent annotation effort).

In the present Section we therefore tried to demonstrate that the training corpora may contain more information than needed for training accurate data–driven dependency parsers.

### 3.3.1 Raising our Hypothesis: Why do we Consider Training Corpora Sizes?

As we show in Sections 3.1 and 3.2, we saw that similar training corpora (i.e., with a similar size in wordforms and a similar distribution of sentences according to their lengths), used to train MaltParser for Spanish, produced models that achieve similar maximum and minimum parsing accuracy values. For this aspect MaltParser shows a stable behavior. But it is important to note that these maximum and minimum values are not always given for the same subsets of text when parsed by all these models. Every subset of text is better parsed by some of these models (usually one or two), while the rest of the models better parse other subsets of text.

Some authors, such as (Nivre et al., 2007) or (Herrera and Gervás, 2008) have also presented signs that the size of the training corpora does not guarantee a high parsing accuracy by itself. A large training corpus statistically permits the presence of a wider range of samples, but equally permits the presence of elements that could induce noise when training samples are not selected one by one. In addition as we show in Sections 3.1 and 3.2, it is proved that the inclusion of new wordforms over a certain threshold does not contribute in the same way to enhance accuracy when training MaltParser for Spanish. These findings suggest that the training corpora of the CoNLL–X Shared Task probably contain information that contributes

less to dependency parsing accuracy when training MaltParser. Taking into account that a parser is trained not only on structure and lexical items, but also on their frequency in the training set, we wanted to analyze if we could eliminate these sentences from the corpora without affecting the performance.

A reduction of the training corpora produces an important reduction in the execution time[4] (training and testing time). Considering that MaltParser is an efficient data–driven dependency parser, as we can observe in Section 2.1.1, it is possible to perform parsing in linear time for projective dependency trees and in quadratic time (in the worst case) for non-projective structures (Bosco et al., 2010), which means that a reduction of N% of the nodes present in the training corpus makes a reduction of N% of execution time in the linear case. If we consider the quadratic case the reduction of N% of the nodes is much more remarkable. When we are considering thousand (even millions) of wordforms this is absolutely significant.

One more important fact when building such kinds of training corpora is their production cost. As an example of the necessary effort for building a dependency annotated corpus, Prokopidis et al. reported in (2005) the process related to the Greek Dependency Treebank (GDT), which was used as the training corpus for Greek in the CoNLL Shared Task 2007. The development of this corpus took a full month's work to 30 annotators to reach a final amount of 70,000 words. Regarding our corpora of interest, if we consider the amount of sentences included in those, we can suppose that the work done to annotate all the training corpora of the CoNLL–X Shared Task that consist in a total amount of 3,521,286 wordforms (considering all the corpora) was hard. Therefore, if we can prove that reduced corpora are a valid way to accurately train dependency parsers, further development processes of such kinds of corpora could be optimized.

In summary, taking into account these previous efforts, our hypothesis is that: *It is possible to create an effective training corpus by removing all those sentences that contain information already present in other samples, when this redundancy, is not useful for the trained system.* By using the word *effective* we mean, a model trained over a reduced corpus that produces accuracy in comparable ranges with the models trained over all the corpora. However, it is also worth noting that apply these insights when starting from nothing is not that easy.

### 3.3.2 Demonstrating our Hypothesis

To analyze the effect of training corpus size on parsing accuracy we incrementally built a training corpus (for each language) and evaluated parsing

[4] A model trained with fewer sentences is faster than a model trained with more sentences

performance for each model trained. Therefore, the experiment is carried out in a similar way as the experiments shown in Sections 3.1 and 3.2.

### Design of the Experiment

We divided every corpus in 15 small subsets. The first 5 using the first 50% wordforms of every corpus and the rest using the second 50% wordforms of every corpus:

This means that for every corpus we created the following subsets:

- The first set containing the first 50% of the wordforms was divided in 5 small subsets, containing each one 10% of the wordforms.

- The second set containing the second 50% of the wordforms was divided in 10 small subsets, containing each one 5% of the wordforms.

Every subset was selected respecting the average sentence length of every whole corpus, so we sampled pseudo randomly. This means that every subset contained (more or less) the same number of wordforms and the average sentence length is, in practical terms, the same for every subset.

Using these 15 small subsets, our experiment was carried out as follows:

- In each iteration we added the next unused subset starting with the corpora that contain the 10% of the corpus, and ending with the last corpora that contain 5% of the corpus, building the whole initial corpus in the last iteration (100%).

- We trained over the resulting incremental corpus and evaluated with the section of the corpus provided as test sets in the CoNLL–X Shared Task.

- We iterated while there were subsets of the corpus that remain unused.

Looking attentively at the evaluation measures to use in this experiment, it is important to take into account that one way to evaluate dependency parsers is to consider parsed texts as sets of wordforms (tokens) and to compute how many tokens are correctly attached. In order to show if the hypothesis holds not only with token based measures and taking into account the fact that MaltParser performs labeled parsing, we must use not only LAS (labeled attachment score) but also other measures. In Appendix B we repeat the same experiment with complete-match measures and we can see that with these measures the improvements when adding more wordforms are even lower. Moreover, see Section 3.4 for a complete experiment of complete-match (or sentence-based) evaluation measures.

### Results of the Experiment

In this Section we show the results of the experiment described above reporting Labeled Attachment Score (LAS) in each iteration.

The results of the experiment for LAS (Labeled Attachment Score) are shown in Table 3.5. Also in Figure 3.5 we show the behavior shown by the models for Labeled Attachment Score (LAS).

| Language | 10% | 20% | 30% | 40% | 50% | 55% | 60% | 65% | 70% | 75% | 80% | 85% | 90% | 95% | 100% |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Arabic | 1.69 | 51.08 | 59.40 | 61.63 | 62.11 | 63.14 | **63.01** | 63.66 | 64.66 | 65.02 | 66.23 | 66.60 | **66.33** | 67.57 | **67.42** |
| Bulgarian | 10.81 | 75.70 | 82.97 | 84.90 | 85.17 | 85.97 | 85.89 | 86.21 | 86.59 | **86.59** | 86.94 | 86.96 | 87.40 | **87.38** | 87.57 |
| Chinese | 62.09 | 77.57 | 81.12 | 82.82 | 84.06 | 84.77 | **84.71** | **84.55** | 85.17 | 85.59 | 85.97 | 86.63 | **86.38** | 86.69 | 86.99 |
| Czech | 0.88 | 71.00 | 72.98 | 74.96 | **74.96** | 75.44 | 75.70 | 76.12 | **75.8** | 76.26 | **76.18** | 76.42 | 76.92 | **76.92** | 77.04 |
| Danish | 11.80 | 77.07 | 80.18 | 81.30 | 81.40 | 82.33 | 82.76 | **82.66** | 83.19 | 83.59 | 83.80 | 83.90 | 84.29 | **84.02** | 84.51 |
| Dutch | 9.10 | 66.05 | 68.40 | 71.57 | 72.19 | 73.35 | 73.90 | **72.97** | **73.79** | **73.63** | **73.43** | 74.73 | **74.73** | 75.01 | **74.47** |
| German | 8.74 | 80.50 | 82.29 | 82.74 | 82.60 | 83.46 | 83.81 | 84.03 | 84.22 | **83.42** | 84.70 | **84.70** | 84.98 | 85.31 | 85.33 |
| Japanese | 18.43 | 87.77 | 89.17 | 89.80 | 90.37 | 90.47 | 90.63 | 91.05 | 91.07 | 91.21 | **91.16** | **91.16** | **90.96** | 91.52 | 91.88 |
| Portuguese | 12.29 | 75.75 | 77.17 | 78.99 | 79.08 | 79.59 | **79.57** | 79.91 | 80.23 | **80.13** | 80.78 | 81.12 | **81.10** | 81.21 | **81.21** |
| Slovene | 4.92 | 49.72 | 54.20 | 58.81 | 59.75 | 61.36 | 62.77 | 63.06 | 64.04 | **63.91** | **64.00** | 64.07 | **63.98** | 64.88 | 65.56 |
| Spanish | 46.36 | 72.25 | 75.98 | 79.30 | **78.96** | 79.78 | 79.86 | **79.64** | 80.22 | 80.75 | **80.36** | **80.75** | 81.55 | **81.39** | 81.87 |
| Swedish | 10.21 | 73.67 | 75.85 | 76.62 | 78.13 | 78.94 | 80.14 | 81.25 | 82.06 | 82.10 | 82.56 | 83.28 | **83.12** | **83.17** | 83.50 |
| Turkish | 12.45 | 58.77 | 61.47 | 61.98 | 62.83 | 63.16 | 63.25 | 63.42 | 63.50 | 63.68 | 63.76 | 64.32 | 64.58 | 64.69 | 64.84 |
| Average | 17.48 | 76.41 | 80.1 | 82.12 | 82.63 | 83.48 | 83.83 | 84.04 | 84.55 | 84.66 | 84.99 | 85.39 | 85.53 | 85.81 | 86.02 |

Table 3.5: General results (LAS) obtained by the iterative models trained with the reduced amount of wordforms corpora. We show in bold the cases in which the result is lower (or the same) as a previous iteration.

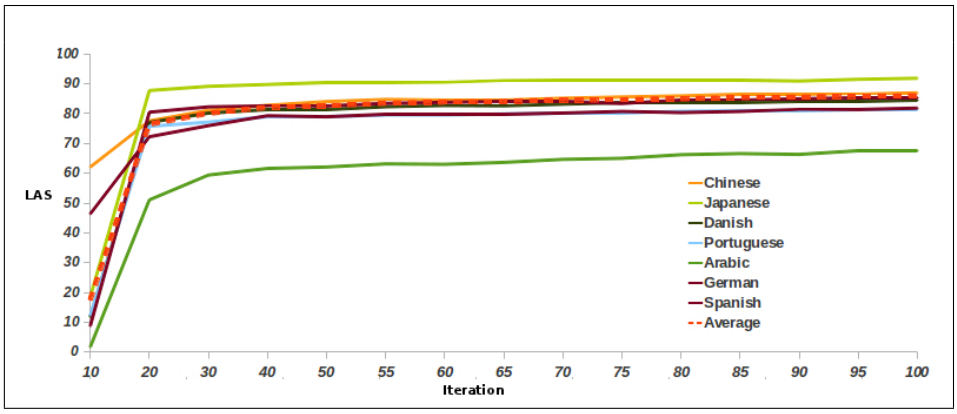



Figure 3.5: Learning curve that all the training corpora show when an iterated training experiment is carried out considering Labeled Attachment Score (LAS).

After systematically studying the accuracy increment given when incrementing the size of the training corpus, we can conclude that over a certain threshold the amount of words in the training corpus affects the accuracy achieved, but it does not provide very high improvements. In other words, a training corpus containing all kinds of sentence lengths does not contribute, in the same way, to parsing accuracy after a given size. So it seems that

"words are not as important as sentences for training corpora". We can conclude that when building training corpora it seems more important to include all kinds of sentence lengths in it rather than giving priority to its total size. Moreover, it seems more important to include all the possible syntactic structures. These findings could lead the community to develop further research on optimizing training corpora, guidelines to develop new corpora or extend existing ones at lower cost.

### Analysis of the Results

As it is shown in Figure 3.5 and Table 3.5, once a significant amount of wordforms (around 50%) has been included, further relative improvements of accuracy by adding more wordforms provide improvements but not that important. As it is shown in Table 2.1 in Section 2.2, an amount of 50% of the sentences varies from 624,704 in the biggest one (Czech) and 14,375 in the smallest one (Slovene). All the data shown in bold in Table 3.5, show the cases in which a model trained over smaller corpora obtained better accuracy (or the same) than the current one for LAS.

Considering the extreme case of the PDT (Czech Prague Dependency Treebank), training a model only over the first 624,704 wordforms (50%), the performance of the trained model achieves 74.96% LAS, while the whole training corpus model trained with the 100% of the corpus, achieves 77.04% LAS. Thus, an improvement of 74.96 points for LAS is given with the first 624,704 wordforms and the subsequent improvement is only 2.08 points for LAS when the other 624,704 wordforms are added. The same thing happens for all languages, taking into account that other languages do not have a corpus as big as the Czech case. Two good examples are Dutch and Japanese, in which we observe an improvement of just a bit more than 0.5% when adding wordforms after the 8th iteration.

Moreover, as it is shown in (Buchholz and Marsi, 2006) test corpora consist of about 5,000 wordforms each, and only words (neither punctuation symbols nor other special wordforms[5]) are considered for measuring the score. An increment or a decrement of 1% LAS, only affects 40 tokens. Thus, comparing the results of model trained with 70% of the training corpus and models trained with the whole training corpus the increment or decrement affects even fewer tokens. However, it is also worth noting that for some languages we always observe substantial improvements when adding more wordforms, such as Arabic, the reason could be simply that the size of the original corpus is not very high, and therefore, it produces that the learning curves for all the languages are not directly comparable.

[5]In this experiment, we are under the set up of the CoNLL-X Shared Task

### 3.3.3 Conclusions

The results of the experiment show that corpora, consisting only on a reduced portion of the original size, result in parsers that suffer a minimal decrease in accuracy. Thus, the presented results should encourage the development of more efficient processes for building training corpora for dependency parsing. We recommend to carefully select the more convenient sentences, according to their syntactic structures. So this study should be understood as a justification for the development of new more efficient processes for building training corpora for dependency parsing.

We believe that future corpora developments should try to avoid including the following:

- A pair of sentences of the same length that share the whole syntactic structure.
- A pair of sentences, one of them shorter than the other one, for which the syntactic structure of the shorter one is completely included in the syntactic structure of the larger one.

Therefore, we could suggest for future corpora developments the following ideas:

- Not to repeat structures, but taking into account that for some languages it is also useful not to repeat substructures.
- To include sentences in the widest possible range of lengths. Also, if the corpus is oriented to train a certain system, it will be useful to consider the behavior of such a system when trained with larger or shorter sentences.

Therefore, the introduction of these guidelines (and similar ones) may lead to great economy effort in the development of new corpora for dependency parsing and/or the extension of existing ones or their adaptations to new domains.

## 3.4 Emphasizing Sentence-Based Evaluation Measures

As we show in Section 1.1.2, the most common evaluation measures are LAS, UAS and LA. These measures were used in the CoNLL shared tasks on Dependency Parsing, and they are token–based. Since these tasks on Dependency Parsing have been very relevant in the area, now this set of measures has become a *de facto* standard when evaluating dependency parsers. Although Yamada and Matsumoto (2003) proposed a sentence–based measure, described in their work as *Complete Rate* measure. Moreover, some recent

works have used complete match measures to evaluate, such as Goldberg and Elhadad (2010) or Nivre (2009).

Therefore, this Section aims to attract attention to sentence–based measures, as a way to get a richer description of the performance of dependency parsers, when needed, combining them with token–based measures. We hypothesized that sentence-based measures may introduce information about issues related to sentence length and training corpora size which is primary linked with the experiments shown in Sections 3.1, 3.2 and 3.3. To this end, we reevaluated the participation of the 19 parsers in the CoNLL–X Shared Task by computing a pair of sentence–based measures over its 13 test corpora.

### 3.4.1 Sentence-Based Measures

We consider parsed texts as sets of sentences. Therefore, we compute measures that can take into account either the whole unlabeled graph (only links between wordforms) or the whole labeled graph (links and labels), for every sentence in the test set. We also consider macro-averaging attachment scores over sentences that seem to be a more informative measure. Since the Shared Task provided labeled parsing, we consider the following evaluation measures:

- Macro–Average LAS (MacroLas) is the percentage of "scoring" tokens in the test set with correct attachment and labeling averaged per sentence.

- Labeled Complete–Match (LCM) is the percentage of sentences in the test set with correct labeled graph.

### 3.4.2 Why do we Think that Sentence-Based Measures Should be Considered?

Nowadays dependency parsers usually show a high overall parsing accuracy when evaluated for LAS, UAS or LA. This means that a high percentage of the processed tokens are correctly linked and/or these links are correctly labeled. But all these tokens pertain to different sentences and generally speaking, only a small percentage of these sentences is actually parsed without any errors. So high values of LAS, UAS and LA mean a high performance from a computational point of view. Nonetheless, the unit of language with proper meaning is the sentence. Then, a human end user eventually would prefer a high percentage of sentences parsed without errors (and a small percentage with several errors), rather than one or two errors for each parsed sentence. Thus, the more sentences without errors the more useful the parser is for a human end user. Under these considerations, sentence–based mea-

sures should be considered in order to add more information to the outcomes provided by the dependency parsers.

Therefore, the reasons given above led us to study the enrichment of token–based evaluation processes with sentence–based measures. This is why we developed the reevaluation described in this Section.

### 3.4.3 Reevaluating the Parsers of the CoNLL–X Shared Task with Sentence–Based Measures

To illustrate our proposal we reevaluated the participation of all CoNLL–X systems[6] computing sentence–based measures. Then, we evaluated each parser by computing MacroLAS and LCM for each test set provided in the Shared Task. The results of this reevaluation are shown in Tables 3.6 and 3.7. The results shown in bold are the best for each corpus.

| Parser | Arab | Bulg | Chin | Czech | Dan | Dutch | Germ | Japa | Port | Slov | Span | Swed | Turk | Tot |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **McD.** | **71.11** | 88.29 | 88.40 | **82.24** | 85.95 | 80.35 | **89.13** | 95.43 | 87.63 | **75.96** | **83.58** | 85.33 | 75.06 | **83.73** |
| Niv. | 70.33 | 88.61 | 89.56 | 79.87 | **86.38** | **80.96** | 88.08 | **96.06** | **88.45** | 71.02 | 82.94 | **86.64** | **76.25** | 83.47 |
| O'N. | 71.06 | **86.63** | 89.50 | 78.74 | 83.95 | 79.16 | 87.87 | 95.42 | 85.69 | 73.91 | 81.87 | 84.50 | 70.23 | 82.19 |
| Che.$^{\uparrow}$ | 69.89 | 87.47 | 87.51 | 78.14 | 83.55 | 74.59 | 86.70 | 95.07 | 85.74 | 73.90 | 81.47 | 83.74 | 73.74 | 81.65 |
| Rie.$^{\downarrow}$ | 70.80 | – | **92.13** | 70.77 | 85.26 | 79.39 | 88.62 | 95.40 | 85.37 | 74.25 | 79.17 | 83.26 | 71.03 | 81.29 |
| Sag.$^{\downarrow}$ | 67.47 | – | 87.60 | 78.83 | 83.99 | 77.73 | 87.19 | 95.28 | 87.06 | 72.84 | 78.40 | 84.45 | 74.60 | 81.29 |
| Cor. | 68.33 | 84.48 | 83.05 | 77.08 | 82.54 | 73.90 | 85.33 | 95.12 | 85.63 | 75.14 | 82.40 | 82.37 | 73.13 | 80.65 |
| Car.$^{\uparrow}$ | 65.72 | 84.23 | 86.76 | 71.62 | 81.07 | 70.34 | 84.33 | 94.18 | 84.13 | 71.04 | 79.20 | 81.40 | 70.36 | 78.80 |
| Cha.$^{\downarrow}$ | 58.56 | – | 87.49 | 69.00 | 81.13 | 75.38 | 86.53 | 94.73 | 82.19 | 71.31 | 80.62 | 84.37 | 71.97 | 78.61 |
| Wu.$^{\uparrow}$ | 67.34 | 81.40 | 78.47 | 54.82 | 79.59 | 73.16 | 79.95 | 95.25 | 82.31 | 70.05 | 73.50 | 75.72 | 67.77 | 75.33 |
| Bic.$^{\uparrow}$ | 58.58 | 80.36 | 80.56 | 66.07 | 76.79 | 72.32 | 76.63 | 92.11 | 76.20 | 66.49 | 73.36 | 77.44 | 66.30 | 74.09 |
| Can. | 53.57 | 79.93 | 83.93 | 56.20 | 79.93 | 77.40 | 81.73 | 93.64 | 74.08 | 57.43 | 68.67 | 81.62 | 65.36 | 73.35 |
| Shi.$^{\uparrow}$ | 67.30 | – | – | – | 76.94 | – | – | – | – | 66.33 | 74.84 | 82.10 | 67.13 | 72.44 |
| Joh.$^{\downarrow}$ | 68.68 | – | 74.29 | 71.50 | 81.87 | 74.59 | 81.17 | 87.26 | 84.01 | 68.15 | 76.39 | 78.51 | 72.82 | 70.71 |
| Liu.$^{\uparrow}$ | 56.66 | 69.00 | 80.00 | 61.31 | 80.34 | 63.91 | 72.60 | 84.68 | 72.28 | 60.12 | 66.49 | 67.96 | 53.17 | 68.34 |
| Yur.$^{\downarrow}$ | 49.36 | 75.04 | 78.09 | 47.31 | 73.50 | 69.30 | 67.85 | 92.17 | 66.49 | 53.27 | 71.01 | 68.96 | 71.85 | 68.02 |
| Sch. | 43.14 | – | 71.66 | 51.47 | 76.87 | 72.44 | 72.26 | 91.59 | 66.55 | 49.00 | 48.34 | 74.52 | 61.98 | 64.99 |
| Dre.$^{\downarrow}$ | 53.95 | 74.56 | 76.36 | 62.91 | 66.78 | 66.36 | 73.34 | 91.09 | 74.63 | 61.53 | 66.88 | 68.76 | 56.47 | 63.83 |
| Att.$^{\downarrow}$ | 50.12 | 70.06 | 51.60 | 55.25 | 64.90 | 49.37 | 66.45 | 44.07 | 72.20 | 56.33 | 65.48 | 63.84 | 44.65 | 58.02 |
| Av | 62.21 | 80.77 | 81.50 | 67.40 | 79.54 | 72.81 | 80.88 | **90.48** | 80.04 | 66.74 | 74.45 | 78.71 | 67.57 | *74.78* |

Table 3.6: Results of the CoNLL–X Shared Task for Macro–Average LAS (MacroLAS). The arrows show the reclassification when considering MacroLAS compared with the LAS results published in the Shared Task.

The results for LCM are normally around 30%, but we must take into account the difficult task that is to annotate sentences that could contain an important number of tokens combined in very different syntactic structures.

MSTParser (McDonald's) and MaltParser (Nivre's) results were really close and the best in the Shared Task. McDonald's parser is the best when considering MacroLAS measure due to the MSTParser accuracy predicting arcs, but Nivre's parser is the best when considering LCM due to the better accuracy predicting dependency labels, as shown in (McDonald and Nivre, 2011). Again, Nivre's and McDonald's systems are the best, and the MacroLAS results demonstrate that they are really accurate when measuring the

[6]Using the outputs published in the CoNLL-X Shared Task website.

| Parser | Arab | Bulg | Chin | Czech | Dan | Dutch | Germ | Japa | Port | Slov | Span | Swed | Turk | Tot |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Niv**$^{\uparrow}$ | 9.59 | **32.91** | 68.05 | 27.12 | **26.09** | **27.46** | 34.73 | **75.32** | **31.60** | 18.41 | **17.96** | **32.13** | 19.26 | **32.36** |
| McD.$^{\downarrow}$ | 9.59 | 30.15 | 62.51 | **27.95** | 24.22 | 25.91 | 34.73 | 72.92 | 23.96 | 18.91 | 17.48 | 27.76 | **19.42** | 30.42 |
| Sag.$^{\uparrow}$ | 8.22 | – | 61.25 | 23.01 | 23.91 | 23.06 | **36.69** | 71.23 | 27.78 | **20.40** | 12.62 | 27.76 | 19.10 | 29.59 |
| Che.$^{\uparrow}$ | 9.59 | 29.15 | 59.63 | 23.01 | 20.19 | 19.43 | 32.49 | 71.51 | 20.83 | 18.91 | 14.08 | 26.48 | 17.50 | 27.91 |
| Rie.$^{\downarrow}$ | 9.59 | – | **72.09** | 13.42 | 21.12 | 22.28 | 32.49 | 71.65 | 21.53 | 13.93 | 10.19 | 23.91 | 13.80 | 27.17 |
| O'N.$^{\downarrow}$ | 9.59 | 26.63 | 62.63 | 20.55 | 18.94 | 21.50 | 31.93 | 71.79 | 21.18 | 15.17 | 11.17 | 25.96 | 13.32 | 26.95 |
| Cor. | **10.27** | 23.87 | 46.83 | 20.82 | 16.15 | 18.65 | 28.85 | 71.79 | 22.57 | 18.16 | 15.05 | 24.16 | 15.25 | 25.57 |
| Cha. | 2.74 | – | 61.59 | 1.64 | 17.39 | 19.95 | 34.17 | 71.51 | 19.10 | 5.72 | 15.05 | 27.25 | 14.44 | 24.21 |
| Car.$^{\uparrow}$ | 8.22 | 20.10 | 58.71 | 17.26 | 15.22 | 17.36 | 25.21 | 67.70 | 19.79 | 14.68 | 15.53 | 20.82 | 13.80 | 24.18 |
| Wu.$^{\uparrow}$ | 8.22 | 23.62 | 47.29 | 0.00 | 13.35 | 17.88 | 24.37 | 72.21 | 21.18 | 13.43 | 7.77 | 14.91 | 11.71 | 21.23 |
| Bic.$^{\uparrow}$ | 8.22 | 13.82 | 43.83 | 11.51 | 10.87 | 18.13 | 17.37 | 62.20 | 4.51 | 7.71 | 9.22 | 16.97 | 10.11 | 18.04 |
| Can. | 0.00 | 14.07 | 46.25 | 0.00 | 12.11 | 18.91 | 22.97 | 65.73 | 0.00 | 0.00 | 4.85 | 18.25 | 10.11 | 16.40 |
| Joh.$^{\downarrow}$ | 8.22 | – | 33.10 | 7.94 | 11.94 | 15.80 | 16.25 | 50.63 | 14.58 | 7.96 | 6.31 | 14.40 | 9.47 | 16.38 |
| Liu.$^{\uparrow}$ | 7.53 | 9.30 | 42.10 | 9.59 | 12.11 | 13.99 | 14.29 | 53.74 | 7.99 | 5.22 | 4.85 | 13.11 | 5.62 | 15.34 |
| Yur. | 0.00 | 10.55 | 44.87 | 0.00 | 9.32 | 16.58 | 12.32 | 63.47 | 0.00 | 0.00 | 5.34 | 11.31 | 14.44 | 14.48 |
| Dre.$^{\downarrow}$ | 0.00 | 5.28 | 39.10 | 8.77 | 0.62 | 14.51 | 12.89 | 59.80 | 6.25 | 5.47 | 2.42 | 7.71 | 4.17 | 12.84 |
| Shi.$^{\uparrow}$ | 8.90 | – | – | – | 12.11 | – | – | – | – | 8.21 | 6.31 | 23.91 | 9.15 | 11.43 |
| Sch.$^{\uparrow}$ | 0.00 | – | 40.72 | 0.00 | 3.73 | 13.73 | 8.12 | 0.28 | 0.00 | 0.00 | 0.00 | 9.51 | 0.00 | 6.34 |
| Att.$^{\downarrow}$ | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Av | 6.24 | 18.42 | 49.48 | 11.81 | 14.18 | 18.06 | 23.33 | **59.64** | 14.60 | 10.12 | 9.27 | 19.28 | 11.61 | *20.04* |

Table 3.7: Results of the CoNLL–X Shared Task for Labeled Complete Match (LCM). The arrows show the reclassification when considering LCM compared with the LAS results published in the Shared Task.

results sentence by sentence. Nevertheless, it seems that, under this perspective, Nivre's parser could be considered a bit better because the differences are wider, more than 2 percentage points, in favour of this parser when considering LCM and the results for MacroLAS are only 0.3 percentage points worse.

Besides that, it is important to remark that the results with MaltParser and MSTParser are similar considering LCM and MacroLAS and they follow a very similar behavior for every language. Therefore, it can be concluded that both types on data–driven dependency parsers are accurate and eligible for parsing complex syntactic purposes. Note that the MacroLAS results are quite similar to the LAS results published in the Shared Task, nonetheless, the parsers that showed better behavior in the Shared Task, obtain much better MacroLAS data and the parsers that showed worse results in the Shared Task obtain much worse results for MacroLAS. It is quite obvious that MacroLAS will yield results close to LAS, since both are averaging the number of correct labeled attachments.

Longer sentences are an interesting issue to tackle because most of the parsers show difficulties parsing them, as mentioned in (McDonald and Nivre, 2011) and in Section 3.2 and 3.3, which means that the results for languages with a longer average sentence length are directly affected by this fact. Most testing data–sets contain sentences of very different lengths, with the exception of Japanese and Chinese, in which the average sentence length is really small and most of the sentences are similar in these terms. Thus, it is also important to take into account that the languages with a shorter average sentence length in the testing data set are the ones with a higher

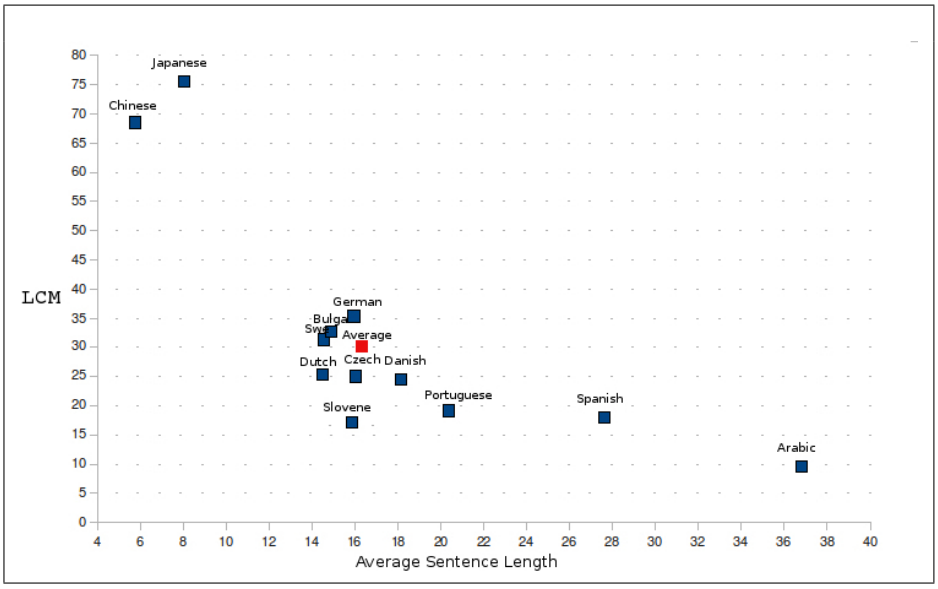



Figure 3.6: Correlation between Average Sentence Length (in the testing data–sets) and the LCM measure when parsed by MaltParser.

LCM after parsing. For instance, the average sentence length for Chinese is 5.78 words and LCM is 49.58. For Arabic, the average sentence length is 36.80 words and LCM is 6.24. In Figure 3.6 we show the correlation between average sentence length and LCM that corroborates this hypothesis considering the correlation between sentence–based measures and the average sentence length. Table 2.1 shows the average sentence length in each corpus.

Besides that, it seems that there are some remarkable differences between models trained with corpora that contain sentences in the same average sentence length, for instance, models trained with the Slovene corpus (18.7 average sentence length) and German corpus (17.8 average sentence length) produced very different results, but it can be explained over the training corpus size of Slovene (29k tokens) and German (700k tokens). Moreover, Czech and German produced similar differences, in this case the Czech corpus is really big (1,249k tokens), but this situation can be explained due to the complexities of the Czech language, such as word–order or irregular grammar, which is a well known issue in dependency parsing.

### 3.4.4 Conclusions of the Study

As shown in subsection 3.4.2 and taking into account the results discussed in subsection 3.4.3, the use of sentence–based measures might give another view on the following question: which dependency parser is better? Considering only token scores the answer may not be enough in some cases, where the

user could want to know whether a Complete–Match accuracy (or close to complete) can be expected or not. This fact have also been evidenced in Appendix B, in which we computed LCM as a control experiment to compare with the one in which we computed LAS.

In summation, it is clear that these measures might be considered when we need a high accuracy per sentence and it is normally needed for a task in which the potential usefulness of dependency parsing is required (see for instance, Chapter 6). We believe that this study shows the importance of sentence accuracy analysis and we would like to encourage researchers to show the results and data considering these measures in order to be able to study the accuracy in a deeper way and taking into consideration all the facts that are involved.

It is worth mentioning that the reclassification of the parsers is wider for LCM than for MacroLAS, as shown in Tables 3.6 and 3.7. Therefore, some parsers have difficulties parsing whole sentences, for instance, Attardi's parser is not able to parse correctly any of the sentences and this knowledge is more than useful when we need to select a parser as a tool to address a task.

Finally, taking into consideration the sentence length factors exposed in the previous subsection, it is also important to make the results directly comparable by building testing data–sets that contain sentences of the same average sentence length and not only containing a similar number of tokens.[7] Moreover, this fact also affects token–based measures because one of the most frequent reasons of errors are due to the dependency length, but it is more evidenced when measuring with LCM, which shows again how sentence–based measures provide non–redundant information.

## 3.5 Chapter Summary

We have shown some experiments and studies that gave us the initial thoughts and motivation to enhance, optimize or apply dependency parsing in a deeper way. Without these initial experiments, the works explained in the subsequent Chapters would have been impossible because this also served us to learn the basis of dependency parsing.

From the initial experiments shown in Section 3.1 and Section 3.2, we saw how the parser behaves and how we can try to modify the possible outcomes, we mainly learned that the quality of the training corpora is very important for the purpose of training. We also learned that long sentences are very useful, they are complex sentences with several nested syntactic structures and it enriches the final accuracy because a transition-based parsing algorithm learns the possible transitions one by one, and this fact makes long sentences very useful.

[7] We had a similar conclusion when we carried out the experiment of Section 3.3

From the experiment shown in Section 3.3, we learned that the training corpora may contain information that could be removed for the purposes of training, therefore, we propose some guidelines to the development of future corpora, by adding new sentences, only if their syntactic structure is not present in others.

From the last study shown in Section 3.4, we saw that giving another perspective may provide information that is not redundant at all, when we want to select the best parser. In particular, we demonstrate that complete-match measures may classify the parsers (according to their accuracy) in a different way. And moreover, they also make apparent the problem of the sentence length in the training and test data sets.

Finally, we emphasize that this chapter provide an interesting point of view on the generated parsers and the possible outcomes.

# Chapter 4

# Enhancing Dependency Analysis

*Progress lies not in enhancing what is,*
*but in advancing toward what will be.*
*Khalil Gibran.*

One of the aims of this thesis is to try to enhance the accuracy of the parsers by showing some contributions and original ideas. In this Chapter we show two different initiatives with this very intention, improving the accuracy of current parsers based on MaltParser:

- The first one (Section 4.1) is a feasibility study for a hybrid combination system in which we only based our studies for the Spanish treebank without modifying internally the parsers. The idea consists in parsing some segments of the sentences separately, more concretely, train specific parsers in order to handle function words that are normally incorrectly parsed by the models trained over the whole corpus.
- The second one (Section 4.2), is based on an in-depth study about the root position during parsing and training time which is something accepted to be at the beginning of the sentences. However, in this work we demonstrate that this problem is indeed relevant and worth studying, because it enhances the accuracy of some parsing algorithms when it is situated at the end (or to the right) of the sentence, or even not having any root at all.

Therefore, this Chapter is actually divided in two different parts in which we show two independent approaches developing ways of enhancing the final accuracy of models generated by a state-of-the-art parser generator: MaltParser.

## 4.1 A Feasibility Study for a Parsing Combination: Towards An N–Version Dependency Parser

We motivate this experiment basing on what we observed in Chapter 3, in which we provide evidence that the size of the training corpus is not necessarily linked to parsing accuracy (Sections 3.1, 3.2 and 3.3) and the alternative perspective by evaluating the parsers with complete-match accuracy (Section 3.4).

To give another example, in CoNLL–2007 MaltParser reached better results for Italian (84.4% LAS), by using a relatively small training corpus of approximately 71,000 wordforms, and for Czech (77.98% LAS), with a training corpus of approximately 432,000 wordforms. Not only MaltParser but all the other systems reached worse results for Czech than for Italian in the CoNLL–2007 Shared Task. It is true, that the Czech free–word order grammar is more complicated but the differences in the size of the training corpora are really wide. So, without an internal modification of the systems, what can we do to make them do their best?

Based on the current results obtained by MaltParser, the aim of the present work is to improve not only the overall results, but to get better complete–match accuracy. Thus, the end user of a dependency parser receives a more satisfactory solution. To do this, we try to find answers to the following questions:

1. Is it viable to improve accuracy by combining small trained different parsers?

2. Which actions to improve accuracy can be automated?

These ideas are also inspired by (McDonald and Nivre, 2007) and (McDonald and Nivre, 2011) and several approaches, like the ones shown in Section 2.3. McDonald and Nivre proposed the combination of two data–driven dependency parsers (MaltParser and MSTParser) using stacking at training time. We propose here an alternative, every parser is trained *a priori* with MaltParser and later, at parsing time, the output of each one is combined to get the final outputs. Note that Nivre arc-eager parser is the best for Spanish parsing, as we can observe in Chapter 5, therefore, the experiments shown concerning this approach are based taking this fact into account. Moreover, this experiment is very linked to the idea of automatically correcting dependency parsing output using hand-crafted or machine-learned correction rules, such as (Anguiano and Candito, 2011) or (Çetinoglu et al., 2011). Which also serves us to motivate what is presented below.

It is worth noting that this first approach remains as a feasibility study (or oracle experiment) because when we tried the real case, the inconsistencies in the treebank made it impossible to make it real. This fact is

explained at the end of this Section. However, we believe that the results of the study shown in this Section, may provide an interesting point of view of the parsing accuracy.

In the following subsections we discuss our proposal on combining different dependency parsers (N parsers), all of them integrated conforming a complete dependency parser. In Section 4.1.1 we motivate the present feasibility study with the complete–match accuracy problem presented in Section 3.4, Section 4.1.2 shows a set of experiments carried out to confirm our hypothesis, Section 4.1.3 shows an intended algorithm developed towards an N–version dependency parser.

### 4.1.1 Motivation

As we saw in Section 3.3, despite a high overall parsing accuracy only (in the case of Spanish) 358 wordforms of the test corpus obtain a 100% LAS, UAS and LA in all parsed sentences, i.e., only 6.3% of the wordforms. When considering sentences, only 38 sentences of the test corpus (18.4% of them) were parsed without errors. That is a 18.4% LCM (labeled Complete Match) when setting the system with the same specifications of the CoNLL–X Shared Task.

We found that there is a small set of words that show an incorrect attachment, labeling or both. These words are the prepositions "a" (*to*), "de" (*of*), " en" (*in*), "con" (*with*), "por" (*for*), the conjunction *and* (which has two wordings: "y" or "e"), and the nexus "que" (*that*). All these words sometimes cause errors in the dependency, in the head tag, or in both tags. For instance, there are only 20 sentences (340 wordforms) in the test corpus with only one error after parsing. That is 9.7% of the corpus sentences and 5.98% of its wordforms. We found that in 10 of these 20 sentences the only failure is caused by one of the words listed above.

The misanalysis of these words could be the reason that the resultant tree might be useless, because these words (such as prepositions, conjunction and nexus) are function words and in many cases the root of subtrees. Therefore, our hypothesis consists in the improvement of the analysis of these specific words towards a better complete–match accuracy and also a better token–based accuracy (global measures, such as labeled attachment scores).

### 4.1.2 Obtaining N Dependency Parsers

The first approach that we studied was a simple n–version parsing model. Our idea was to determine whether some kinds of "difficult" words could succesfully be parsed by specific parsers while a general parser would parse the rest. Therefore, the N–version dependency parser works as follows: the general dependency parser parses the whole sentence. If a pattern is detected, the specific dependency parser associated to that pattern parses

the sentence in the same way that the general parser did but taking into account the expected behavior. Finally, an algorithm would swap the node obtained for the specific word, and the rest of the dependency-parsed tree given by the specific dependency parser will be ignored. In this way, we are obtaining a better dependency-parsed tree for the sentence.

First, we did an in–depth study of each one of the words listed in the Section 4.1.1. This study consisted of finding out the set of different cases in which each word could be attached and labeled, and training a specific parser for each case found. As a result, we found that the conjunction is the word that caused a parsing error more frequently. This is why we selected it as a first case to study in order to determine if these kind of techniques are feasible to improve parsing accuracy. The cases studied are shown in Table 4.1.

The study of the errors given when parsing conjunctions began with a manual analysis of the Spanish treebank. Therefore, we extracted automatically from the corpus every sentence containing a conjunction ("y" or "e" in Spanish) resulting in a total of 1,586 sentences with at least one conjunction. We inspected these sentences to find labeling patterns. This way we obtained a list of patterns that depend on the conjunction action. For instance, a pattern is given when the conjunction acts as a nexus in a coordinated copulative sentence, and another pattern is given when it acts as the last nexus in a list of nouns. In the following sentence: *Los activos en divisas en poder del Banco Central* **y** *el Ministerio de Finanzas se calculan en dólares estadounidenses* **y** *su valor depende del cambio oficial rublo–dólar que establece el Banco Central (The foreign exchange assets held by the Central Bank and the Ministry of Finance are calculated in U.S. dollars and its value depends on the official ruble–dollar exchange rate established by the Central Bank)* the first *y* is a nexus between the proper nouns *Banco Central (Central Bank)* and *Ministerio de Finanzas (Ministry of Finance)* and the second *y* acts as a coordinated copulative nexus. These patterns guided the approaches described below. Moreover, we developed a similar study for the rest of function words that are listed in Section 4.1.1.

The first specific parser that we tried to obtain was supposed to parse quoted sentence sections containing conjunctions, which is shown first in this Section. This situation is quite common and corresponds to one of the labeling patterns that we identified as problematic. We then show the whole approach, which consists of an in–depth study of all the words that are more frequently incorrectly parsed, as a way of studying the feasibility of this idea. The study consists of finding the different ways that these words are attached and labeled, and training a specific model for each case, by building automatically training corpora for each case.

### First Semi-Automatic Approach

In this first approach we show our works towards an n–version dependency parser. We trained a specific model for coordinated copulative sentences to check the feasibility of building n specific parsers in the way of achieving higher accuracy. To this end, we automatically extract a subset from the training corpus with the set of unambiguous coordinated copulative sentences contained in the section of the Spanish treebank that was provided as training corpus in the CoNLL–X Shared Task. This specific training corpus contains 361 sentences (10,561 wordforms). Then, we parsed all the coordinated copulative sentences contained in the section of the treebank that was provided as test corpus in the CoNLL–X Shared Task (16 sentences, 549 wordforms). We set up the experiments described above with the same feature model, that Nivre's group used in its participation in the CoNLL–X Shared Task. We found that the conjunction was incorrectly parsed 8 times (in a test set containing 16 conjunctions). This fact led us to investigate with different feature models. After a few failed attempts we found a feature model in which 12 of the 16 conjunctions were parsed correctly, by extending the stack window of part-of-speech and adding some features at the FEATS window over the stack and the buffer.

Despite the results being enhanced by using the new feature model, the general parsing model parses 13 of these 16 conjunctions without errors. It could mean that specific models are not feasible for our objectives.

### Definitive Semi-Automatic Approach: N Parsers

Since the accuracies reached by both models in the first approach were very similar, we developed some other experiments to confirm or reject our hypothesis. Thus, we tried new specific parsers for other combinations of all the function words that we took into account. We created a parser for each specific pattern found for each concrete word. Once the sentence is parsed with the specific model, the result for the "problematic" word is replaced in the resulting tree obtained by the general model. The labeling given to each word by the specific parser is cut from this parsing and pasted into the parsing given by the general model, by replacing the labeling given to these words by the general parser. This easy solution is possible because these words can be changed without affecting the rest of the parsing and it therefore does not produce inconsistencies, because they are function words that are normally the root of different subtrees. This is why the final system is semi-automatic.

The results obtained for all these words are shown in Table 4.1. They are normally better when using a specific parser instead of the general parser, with just some exceptions. But sometimes the specific parsers reach the same accuracy as the general parser, so it does not make sense to use the

specific parser in such cases. For instance, when parsing the word *de* when attached to an adjective or an adverb, both the general parser and the specific parser show 100% LAS. Only when the word $y$ (or $e$) acts as a nexus in coordinated copulative sentences we could not find a specific parser better than the general parser (the general parser reaches 81.3% $\text{LAS}_{y/e}$ and the specific parser reaches 75% $\text{LAS}_{y/e}$). In 21 of the 28 cases identified it is better to use the specific parsers.

| | | **Case** | | | | | |
|---|---|---|---|---|---|---|---|
| **Word** | | #1 | #2 | #3 | #4 | #5 | #6 |
| y/e | **Label** | – | – | – | – | | |
| | **Attached to a** | verb$^{\leftarrow}$ | proper noun$^{\leftarrow}$ | common noun$^{\leftarrow}$ | adjective$^{\leftarrow}$ | | |
| | **LAS$_{y/e}$ original** | ***81.3***% | 80% | 66.7% | 80% | | |
| | **LAS$_{y/e}$ combined** | 75% | ***100***% | ***80***% | ***100***% | | |
| a | **Label** | CD | CI | CC | CREG | – | – |
| | **Attached to a** | verb$^{\leftarrow}$ | | | | | noun$^{\leftarrow}$ |
| | **LAS$_a$ original** | 62.5% | 42.9% | 60% | 25% | ***0***% | 50% |
| | **LAS$_a$ combined** | ***87.5***% | ***100***% | ***100***% | ***75***% | ***0***% | ***100***% |
| de | **Label** | CC | CREG | – | – | | |
| | **Attached to a** | verb$^{\leftarrow}$ | | adverb$^{\leftarrow}$ adjective$^{\leftarrow}$ | noun$^{\leftarrow}$ | | |
| | **LAS$_{de}$ original** | 0% | 0% | ***100***% | 83.3% | | |
| | **LAS$_{de}$ combined** | ***100***% | ***100***% | ***100***% | ***96.7***% | | |
| que | **Label** | SUJ | – | SUJ | | | |
| | **Attached to a** | verb$^{\rightarrow}$ | | verb$^{\leftarrow}$ | | | |
| | **LAS$_{que}$ original** | 88.5% | 86.4% | 0% | | | |
| | **LAS$_{que}$ combined** | ***92.3***% | ***95.5***% | ***100***% | | | |
| en | **Label** | CC | CC | CREG | – | | |
| | **Attached to a** | verb$^{\rightarrow}$ | verb$^{\leftarrow}$ | | noun$^{\leftarrow}$ | | |
| | **LAS$_{en}$ original** | ***83.3***% | 92.6% | 50% | 62.5% | | |
| | **LAS$_{en}$ combined** | ***83.3***% | ***100***% | ***100***% | ***87.5***% | | |
| con | **Label** | CC | CREG | – | – | | |
| | **Attached to a** | verb$^{\leftarrow}$ | | | noun$^{\leftarrow}$ | | |
| | **LAS$_{con}$ original** | 60% | 40% | ***100***% | 66.7% | | |
| | **LAS$_{con}$ combined** | ***80***% | ***100***% | ***100***% | ***83.3***% | | |
| por | **Label** | – | CAG | CAG | | | |
| | **Attached to a** | noun$^{\leftarrow}$ | comma$^{\leftarrow}$ | adjective$^{\leftarrow}$ | | | |
| | **LAS$_{por}$ original** | ***100***% | ***100***% | 80% | | | |
| | **LAS$_{por}$ combined** | ***100***% | ***100***% | ***100***% | | | |

Table 4.1: Attachment and labeling for all the studied words in the Spanish treebank. Specific LAS for each word and case, before and after the application of our method. The left arrow ($^{\leftarrow}$) after a part of speech indicates that this part of speech is before the considered word in the sentence. The right arrow ($^{\rightarrow}$) indicates that the part of speech is after the word.

Afterwards, we recomputed LAS, UAS and LA for this combined parsing, which are better than those obtained with the general parsers, as is shown in Table 4.1. It means a slight enhancement with respect to the results given by the general parsing model. In addition, in the combined parsing these words do not belong to the set of words that are most frequently incorrectly

parsed. This improvement seems to indicate that this n–version parsing model is feasible and overall accuracy could be substantially improved.

In some cases the given improvement seems spectacular. For instance, when parsing the word *de* when attached to a verb, the general parser shows 0% LAS and the specific parsers show 100% LAS. It is due to the small amount of samples present in the test corpus. For instance, if the test set contains only one sample for a specific case and this sample is correctly parsed, then we obtain 100% LAS. But it does not mean that the parser will parse every given sample of this case with 100% LAS. For the given example the test corpus contained only 4 samples. All these samples were wrongly parsed by the general parser but perfectly parsed by the two specific parsers involved. Therefore LAS was enhanced from 0% to 100%, but this is for the given test corpus. If the test corpus contained more samples, perhaps the specific parsers might not have reached 100% LAS. Usually the local improvement obtained by the specific parsers is very high, but as said before it must be taken cautiously because of the small amount of samples in some of the cases in our test corpus, which are usually between 2 and 10 for each case, 30 being the maximum. Nevertheless, parsing accuracy is reasonably homogeneous and similar accuracies should be expected even when increasing the number of samples in the test set.

In addition, we found that the word *de* attached to a verb with the undefined label "–" is a given case in the training corpus that is not given in the test corpus. Of course, for this situation no error is given by the general parser, but, how can we know if the parser can tackle such a case if it is not present in the test corpus? This is why, if we want to obtain a high performing parser, we must carefully rebuild the train and test corpora.

#### Outcomes of the Semi-Automatic Experiments

The use of specific parsers can improve the complete–match accuracy. And this results to the improvement of the overall accuracy. Dependency parsers can be useful for human end-users, who would presumably use such parsers to analyze small sections of text. In this case, a single error in the parsing of one sentence is significant. This is because the developers of dependency parsers should care about high complete–match accuracy. After parsing the test corpus with our semi-automatic n-version parser we found that 42 (20.3% LCM) of the parsed sentences showed no parsing errors, while 38 (18.4% LCM) of them were perfectly parsed with the general parser. This improvement of the complete–match accuracy has not only a better experience for human end-users but also an improvement of overall accuracy. When parsing the test corpus by combining the action of the general parser and our proposed specific parsers, we obtained the following results for overall accuracy: 82.68% LAS, 85.73% UAS and 90.84% LA. This means an improvement of 1.38% LAS, 1.06% UAS and 0.78% LA in overall accuracy

with respect to the results of the general parser alone.

Therefore, our proposed idea seems a way to improve parsing accuracy by systematically avoiding the errors given by a general parser. This improvement is greater when eliminating the errors caused by a frequent word, as shown in Figure 4.1. The set of bars shows the increments of LAS, UAS and LA when cumulatively adding the action of specific parsers for each word considered. The first word for which we added the action of its specific parsers was the conjunction (*y* o *e*), because the conjunction is the word most frequently parsed wrongly by the general parser. Following this idea, we cumulatively added the action of specific parsers for each of the words considered, starting with those that caused most parsing errors when using the general parser. In the end, when adding the action of specific parsers for the word *por*, we obtained the action in synergy for all the specific parsers listed in Table 4.1 and the general parser. We can observe in Figure 4.1 that LAS, UAS and LA increased notably when adding the action of specific parsers for the conjunction and the preposition *a*. Nonetheless, LA did not increase when we added the action of a specific parser for the conjunction, but this is because the general parser does not fail when attaching the conjunction to other words (it only fails in the labeling of the conjunction).

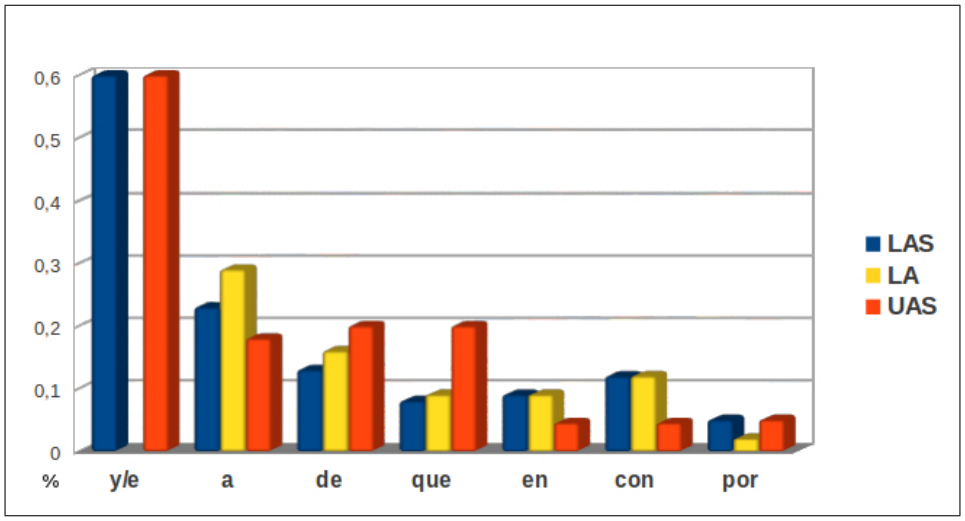



Figure 4.1: Increments of overall LAS, UAS and LA due to the action of specific parsers that avoid the most frequent errors, given by certain words.

As expected, the more infrequent the word that causes parsing errors is, the less the contribution of its specific parsers to the overall action is. So the effort of building specific parsers may not be worthwhile, given the improvement obtained. It is worth mentioning that the conjunction causes 56 parsing errors with the general parser, *a* causes 48 errors, *de* 44 errors, *que* 42 errors, *en* 37 errors, *con* 17 errors and *por* 16 errors. Also, the

increments obtained are not regular and this is because of the number of samples of each case considered present in the test corpus and the accuracy of their specific parsers.

Although a large percentage of the most frequent errors given when parsing with the general parser are eliminated with the n–version parser, a remarkable amount of errors remains. Since specific parsers have been developed for only a small set of words, some other words remain without a specific treatment and cause errors. This means that a great effort is needed. A lot of errors could be avoided by implementing more complex n–version parsers, covering a bigger amount of "difficult" words than the ones presented here. But some other errors could be inherent to the implementation of MaltParser and cannot be avoided. Also, as suggested in Chapter 3 (Sections 3.1, 3.2 and 3.3), some other errors could be treated by carefully building the training corpora.

### 4.1.3 The Intended Automatic N–Version Dependency Parser Algorithm

Once we concluded that the combination of several specific parsers could be a feasible technique to enhance parsing accuracy, the following step was to develop it automatically – that is, the algorithm that makes all the specific parsers work in synergy.

Our approach for the algorithm that sends each different wordform to the most appropriate specific parser is based on pattern matching and rules. So when a certain pattern is recognized in the sentence to be parsed, it is sent to the most suitable specific parser by means of a rule. This algorithm were only implemented for the preposition "a" and the conjunction, which in Spanish has two wordings: "e" and "y", because these two are the words most frequently parsed incorrectly by the general parser. It works as follows:

1. A sentence is parsed with the general parser. In Figure 4.2 we show how the general parser parses the sentence: *Trasladó el material a Madrid* [*he (or she) moved the material to Madrid*], we can observe that the general parser makes an error in the node containing the preposition "a". The correct attachment for this node must be the main action of the sentence: *Trasladó*, and the correct label is "CC" that is the adjunct of the verb. The general parser produces an error in both things.

2. If the algorithm detects that there is a conjunction or a preposition "a" in the sentence, it sends the whole sentence to the most appropriate specific parser. Each parser is set with different specifications, shown in Table 4.1.

3. The selected specific parser parses the sentence. In Figure 4.3 we

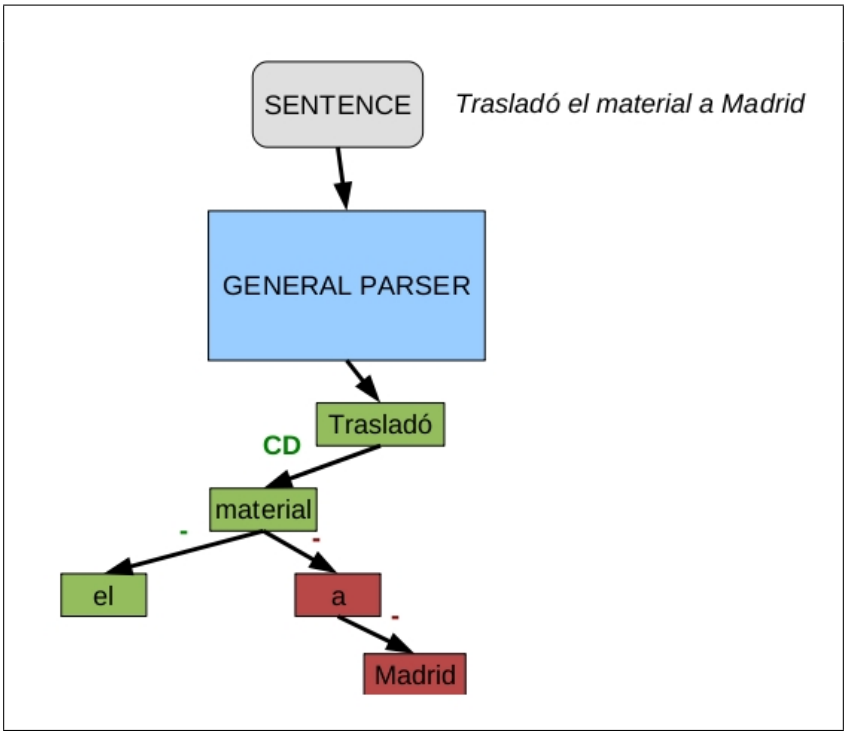



Figure 4.2: The General Parser parsing the sentence: *Trasladó el material a Madrid* [*he (or she) moved the material to Madrid*].

can observe how the specific parser parses the same sentence as the general parser but does not make the same error as the general parser. In this case, the specific parser correctly parses the node containing the preposition "a" (it does not correctly parse other sections of the sentence nonetheless).

4. The algorithm removes the node containing the conjunction or the preposition "a" from the tree returned by the general parser.

5. The algorithm inserts the node containing the conjunction or the preposition "a", produced by the specific parser into the general parsed tree. In Figure 4.4 we can observe how the algorithm inserts the node containing the preposition "a" into the tree given by the general parser. When inserting the node, the algorithm also inserts its subtree.

6. The algorithm returns the whole dependency tree that is produced from the suitable combination of the dependency trees given by the general and the specific parsers.

By applying the algorithm described we find that our n–version system incorrectly parsed 55 conjunctions while the general parser by itself parsed

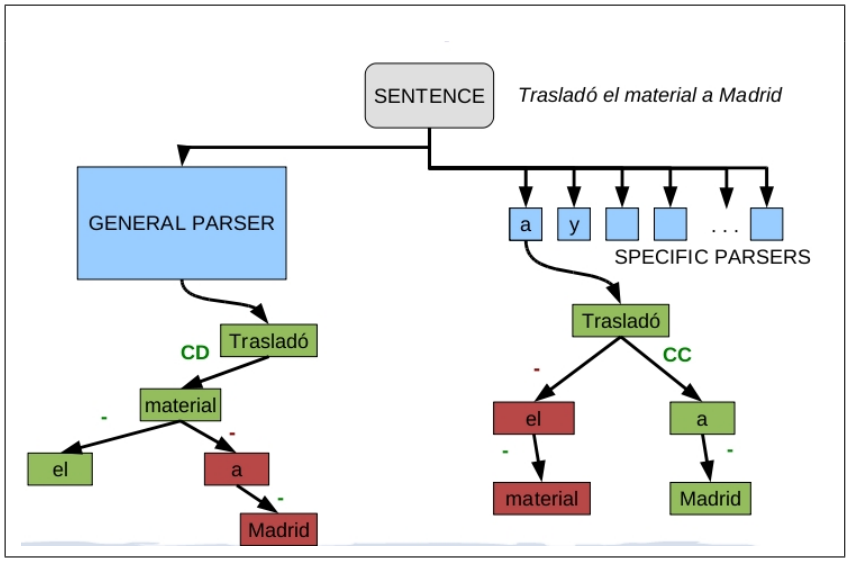



Figure 4.3: The Specific Parser parsing the sentence: *Trasladó el material a Madrid* [*he (or she) moved the material to Madrid*].

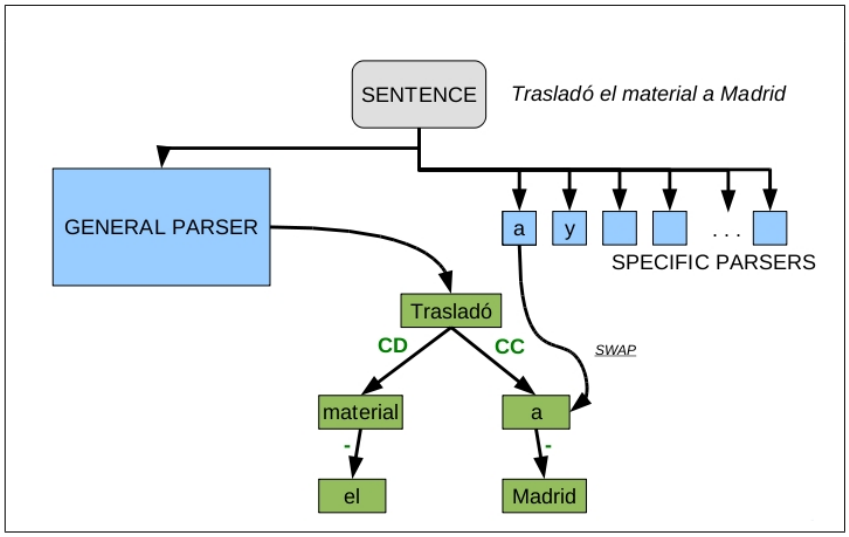



Figure 4.4: Swapping the node containing the preposition 'a' given by both parsers.

56 conjunctions incorrectly. For the preposition "a" we got similar results: our n–version system incorrectly parsed 50 prepositions and the general one parsed 48 prepositions incorrectly. Unfortunately, the results are not as good as the ones shown in the semi-automatic approach shown in Section 4.1.2.

These negative results do not mean that the n–version technique should

be rejected. An explanation for this problem can be found in the corpus. This corpus was built automatically with a strong linguistic and manual validation step, but we have identified some errors that still remain in the corpus. For instance, we realized that two sentences with the same syntactic structure can be found in the Spanish treebank with different annotations for the words studied.

Thus, a revision of the given sentences is necessary to evaluate our n–version model properly, because we do not have a 100% error free corpus for Spanish to compare. Nonetheless, it is also true that having a corpus with less errors will produce better results for the general parser trained with the whole corpus.

### 4.1.4 Conclusions

As a first conclusion and considering the feasibility study, we certainly think that it seems worth trying to improve the accuracy focusing on the words that are usually incorrectly parsed. Moreover, considering that these words are usually function words that are the root of dependency subtrees, it is obvious that improving the accuracy of them, the accuracy of the final parsing will be increased. Moreover, it seems that focusing on the root of trees (or subtrees, like in this experiment) seems an interesting way of enhancing the accuracy and providing an interesting study about how and where the parser fails.

However, due to the fact that this experiment is just a feasibility study we conclude that the corpora must have a consistent annotation in order to carry out this kind of experiment shown in this Section in an automatic way, it seems not acceptable that sentences with the same syntactic structure are annotated in a different way. This fact leads to a similar conclusion as the one that we have in Chapter 3, we should encourage the annotation of the corpora double-checking the final annotation and selecting the sentences very carefully.

## 4.2 Enhancing a Transition-Based Parsing Algorithm by Modifying the Root Position

As we saw in Section 2.1.1, a transition-based dependency parsing algorithm normally makes use of two different data structures: the buffer and the stack. The buffer contains all the tokens (or wordforms) that belong to the sentence. The stack stores the nodes that are susceptible of being attached during the parsing transitions and the algorithm decides what to do according to what it finds in these data structures, normally at the top of the stack and the following token that is coming out the buffer. See Section 2.1.1.

The root node, which is the one that permits to have dependency tree

structures forcing just a single root for each parsed sentence, is normally stored in the first parsing step at the top of the stack. In this way the parsing algorithm generates attachments (normally just one) to this root node. Therefore, in this scenario, the root node is located at the beginning of the sentence as a "dummy" extra word. According to this, it has been stated in several books published, such as (Nivre, 2006) and (Kübler, McDonald, and Nivre, 2009), that the position of this artificial node during parsing time is not relevant. And it is just a matter of definition that does not affect parsing performance. Is this fact really true?

There are several transition-based parsing algorithms that traverse the sentence from left to right,[1] and they could be grouped in several families according to parsing strategies. In this Section we are focusing mainly in two that behave differently: the arc-eager and the arc-standard (Nivre, 2003; Nivre, 2008), both are implemented in MaltParser (Nivre, Hall, and Nilsson, 2006). These two parsing strategies differ principally from the way they generate left attachments from the incoming nodes from the buffer to the nodes that are encountered at the top of the stack. The arc-eager parsing algorithm is greedy in the way that as soon as it can, it generates left attachments. However, the arc-standard parsing algorithm behaves in a different way, being more lazy when left attachments could be generated. The position of the root node seems then relevant, because the attachments to the root node are basically left attachments following the original idea.

It has been observed that in a multi-clause sentence we usually find several post-subordinate clauses but it is extremely uncommon to have more than one pre-subordinate clause. This fact may invite to think that an artificial root node located at the beginning (or to the left) of the sentence as a starting point for parsing may produce a drop in the accuracy getting spurious arcs or incorrect root attachments to this root node when the arcs to this artificial root node are being generated.

Taking all of the above into account and knowing the fact that it has been observed that MaltParser with arc-eager parsing order tends to have low precision for attachments to the root (Nivre, 2008), we therefore raise the following hypothesis: *The low precision for attachments to the root in the Nivre arc-eager algorithm is actually caused by the interaction of the arc-eager strategy and the position of the root at the beginning of the sentence, and more importantly, the situation of the root node during parsing really matters and affects the performance in different scenarios.*

In this Section, we present several experiments forcing the parser to behave differently by modifying the root position during parsing time in different scenarios. We report results, raise new hypotheses and explanations according to the selected parsing algorithm that corroborate our initial hypothesis and different ideas that we introduce during the discussion.

[1] See Section 2.1.1 to find a description of each family of parsing algorithms.

### 4.2.1 Root to the Left versus Root to the Right

In order to corroborate the hypothesis raised above, we carried out the following experiment: we run for all the corpora from the CoNLL-X Shared Task (Buchholz and Marsi, 2006) the Nivre arc-eager algorithm in the scenarios shown below:

- Root to the left: Nivre arc-eager with pseudo-projective parsing and default settings. Therefore, the parser adds in the first parsing step the root node at the top of the stack, which in a way is the same as having the root at the beginning (or to the left) of the sentence.
- Root to the right: we modified the corpora adding an extra root node located at the end of the sentence. All the root attachments are previously modified and instead attached to this extra root node. After parsing, we automatically processed the sentence from left to right changing the attachments of the extra node to the real root node. Moreover we run the parser without root during parsing,[2] which means that the real root is the one added when we modified the data.

In all of our experiments we report results with pseudo-projective parsing (Nivre and Nilsson, 2005), with the intention of avoiding the noise produced by non-projective arcs. We therefore force the parser to generate the existing non-projective arcs in a post-processing step.

The results for LAS and UAS are shown in Table 4.2. The corpora is sorted in the Table according to the percentage of scoring tokens with HEAD to the right in an incremental order.[3] See Section 2.2 to find the percentage of left and right attachments for the corpora of the CoNLL Shared Tasks.[4]

As we may observe in the results shown in Table 4.2, the results provided by the parser when the root node is located at the end of the sentence (to the right) produced an important improvement in the accuracy for most of the languages. In fact, it provided better results than the first scenario in all the languages for LAS and in 12 of the 13 languages for UAS (the exception is Dutch, which provided a non-significant improvement for LAS). We provide answers to this in the following sections.

### 4.2.2 An In-depth Experiment

In order to be sure about the results and outcomes obtained in Section 4.2.1, we present three different scenarios in which the root node has a different role during parsing. We perform the experiments by using MaltParser, modifying the training corpora and using the root options that MaltParser contains.

[2] This is indeed possible by using the options provided by MaltParser, see http://maltparser.org

[3] All the tables of the present Section are sorted following the same idea.

[4] You may also visit http://ilk.uvt.nl/conll/paper_submission.html#table

| Corpora | Root-Pos | LAS | UAS |
|---|---|---|---|
| Arabic | Left | 63.71 | 74.53 |
| | Right | **64.15** | **74.97** |
| Danish | Left | 80.74 | 86.71 |
| | Right | **82.38** | **87.94** |
| Bulgarian | Left | 84.64 | 89.81 |
| | Right | **85.76** | **90.78** |
| Spanish | Left | 78.14 | 82.15 |
| | Right | **78.64** | **82.49** |
| Portuguese | Left | 83.71 | 88.36 |
| | Right | **84.17** | **88.62** |
| Swedish | Left | 83.19 | 89.34 |
| | Right | **83.59** | **89.70** |
| Czech | Left | 72.94 | 80.16 |
| | Right | **73.96** | **81.16** |
| German | Left | 83.27 | 86.10 |
| | Right | **83.93** | **86.72** |
| Slovene | Left | 67.75 | 77.84 |
| | Right | **69.98** | **79.62** |
| Dutch | Left | 70.95 | **74.55** |
| | Right | **71.05** | 74.51 |
| Chinese | Left | 84.55 | 89.07 |
| | Right | **85.15** | **89.70** |
| Japanese | Left | 88.71 | 91.25 |
| | Right | **89.77** | **92.12** |
| Turkish | Left | 56.54 | 71.78 |
| | Right | **56.64** | **72.16** |

Table 4.2: Nivre arc-eager results for the two scenarios showing labeled and unlabeled attachment scores.

- Scenario 1 (SC-1): Non existence of a root during parsing. The parser then attach all the nodes without head to a dummy root node with a default label 'ROOT' when there are no more transitions to make. This can be achieved by using the "allow root" option to false, which means that there is no root during the parsing transitions.

- Scenario 2 (SC-2): Root to the right of the sentence. We modified the corpora adding an extra root node located at the end of the sentence. All the root attachments are modified previously and attached instead to this extra root node. After parsing, we automatically post-process the sentence from left to right changing the attachments to the extra node to the root node.

- Scenario 3 (SC-3): Root to the left of the sentence. The same thing that we did in the second scenario, but adding an extra root node located at the beginning of the sentence.

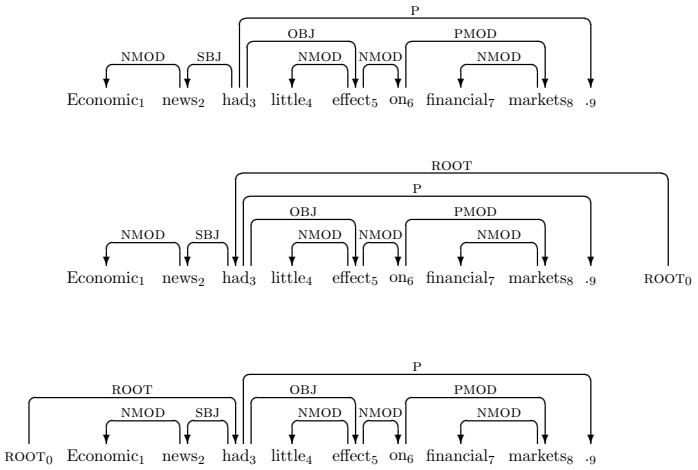



Figure 4.5: Dependency graphs of types None (SC-1), Right (SC-2) and Left (SC-3) for an English sentence extracted from the Penn Treebank.

Figures 4.5 and 4.6 illustrate the three types of dependency graphs with examples taken from the Penn Treebank of English, and the Prague Dependency Treebank of Czech. In the former case, it is assumed that the dummy root node always has exactly one child, with a dummy dependency label ROOT. In the latter case, the dummy root node may have several children and these children have informative root labels indicating their function (Pred and AuxK in the example). Note also that the Czech dependency graph of type None is not a tree, but a forest, since it consists of two disjoint trees.

In order to corroborate our hypotheses we carried out the experiment with the Nivre arc-eager algorithm and the Nivre arc-standard algorithm.

### Nivre arc-eager

In this control and more detailed experiment, we raise the hypotheses shown below.

- For SC-2 and SC-3, we expect similar outcomes as the ones that we observe in the two scenarios studied in the the experiment shown in Section 4.2.1.
- For SC-1 we expect a similar behavior as the one that we have in SC-2, with a significant lack of labeled accuracy in the languages with multiple root labels (Arabic, Czech, Slovene and Portuguese) because the parser will instead use the dummy label 'ROOT'. However, we

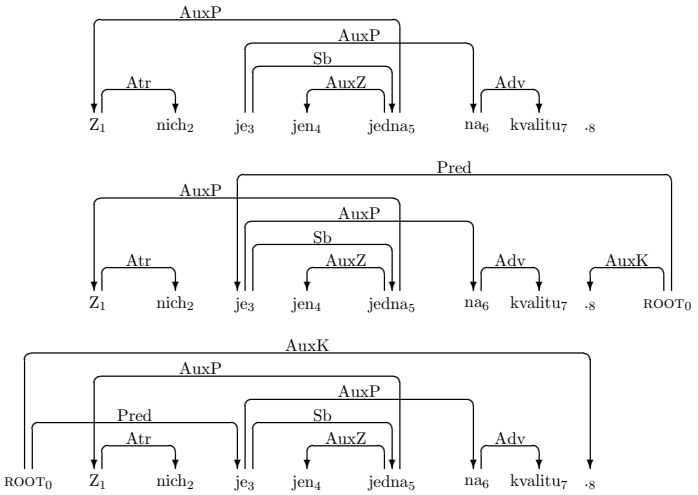



Figure 4.6: Dependency graphs of types None (SC-1), Right (SC-2) and Left (SC-3) for a Czech sentence taken from the Prague Dependency Treebank. Gloss: Z/Out-of nich/them je/is jen/only jedna/one-FEM-SG na/to kvalitu/quality ./. = "Only one of them concerns quality."

also expect that SC-2 is a little bit more accurate because by locating the root to the right the way the parser attach to the root in the last steps of the parsing process is been improved, and, more important, the noise is been avoided in the rest of attachments.

The results of the experiments are shown in Table 4.3. As we can observe in SC-3 the root node is constantly "in play" while in SC-2 it only shows up at the end. We believe then that there is one additional factor to consider, namely the fact that the right-arc transition is arc-eager, meaning that it has to attach to the left as early as possible, which causes spurious attachments. By contrast, the left-arc transition cannot be applied until the entire subtree of the dependent has been built, and this delay probably blocks some spurious attachments.

In the model with the root to the left (SC-3), we could expect (in an ideal case) the same behavior as we have in SC-2, but certainly we are not getting the same. We observe a drop in labeled and unlabeled attachment scores, however we observe a normally higher precision for root attachments and lower recall. This could be explained due to the fact that the algorithm has to make attachments to the "artificial root" node that is to the left.

This is also why SC-3 produces a significant under production of attachments to the right (compared to the other 2 and the expected proportion) in the corpora that show a high proportion of attachments to the left (Ara-

| | | Root Attachments | | | Left Attachments | | | Right Attachments | | | LAS and UAS | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Corpora | Root-Pos | # | Recall | Precision | # | Recall | Precision | # | Recall | Precision | LAS | UAS |
| Arabic | No Root (SC-1) | 315 | 74.24 | 69.52 | 4206 | 95.93 | 94.82 | 469 | 61.52 | 70.58 | 60.00 | **75.17** |
| | Right (SC-2) | 318 | 73.56 | 68.24 | 4199 | 95.81 | 94.86 | 473 | 61.71 | 70.19 | **64.15** | 74.97 |
| | Left (SC-3) | 339 | 84.75 | 73.75 | 4252 | 96.87 | 94.71 | 399 | 57.06 | 76.94 | 63.63 | 74.57 |
| Danish | No Root (SC-1) | 335 | 91.33 | 88.06 | 3715 | 97.63 | 97.42 | 960 | 90.71 | 92.60 | 82.36 | 87.88 |
| | Right (SC-2) | 335 | 91.64 | 88.36 | 3715 | 97.65 | 97.44 | 960 | 90.82 | 92.71 | **82.38** | **87.94** |
| | Left (SC-3) | 341 | 86.69 | 82.11 | 3749 | 97.92 | 96.83 | 920 | 87.45 | 93.15 | 80.60 | 86.59 |
| Bulgarian | No Root (SC-1) | 413 | 94.22 | 90.80 | 3205 | 98.19 | 97.91 | 1395 | 95.00 | 96.63 | **85.76** | **90.80** |
| | Right (SC-2) | 414 | 94.22 | 90.58 | 3204 | 98.15 | 97.91 | 1395 | 95.00 | 96.63 | **85.76** | 90.78 |
| | Left (SC-3) | 410 | 90.20 | 87.56 | 3239 | 98.37 | 97.07 | 1364 | 93.02 | 96.77 | 84.64 | 89.83 |
| Spanish | No Root (SC-1) | 215 | 83.25 | 76.28 | 3057 | 96.66 | 95.75 | 1719 | 92.41 | 94.94 | **78.64** | **82.51** |
| | Right (SC-2) | 216 | 83.76 | 76.39 | 3054 | 96.63 | 95.81 | 1721 | 92.53 | 94.94 | **78.64** | 82.49 |
| | Left (SC-3) | 214 | 79.70 | 73.36 | 3070 | 96.70 | 95.37 | 1707 | 91.90 | 95.08 | 78.14 | 82.15 |
| Portuguese | No Root (SC-1) | 309 | 92.01 | 85.76 | 3022 | 98.40 | 97.88 | 1678 | 96.09 | 98.21 | 79.12 | 88.60 |
| | Right (SC-2) | 309 | 92.01 | 85.76 | 3022 | 98.40 | 97.88 | 1678 | 96.09 | 98.21 | **84.17** | **88.62** |
| | Left (SC-3) | 293 | 87.85 | 86.35 | 3037 | 98.54 | 97.53 | 1679 | 95.98 | 98.03 | 83.77 | 88.36 |
| Swedish | No Root (SC-1) | 403 | 93.32 | 90.07 | 2741 | 96.68 | 96.83 | 1877 | 95.39 | 95.90 | 83.49 | 89.60 |
| | Right (SC-2) | 403 | 93.32 | 90.07 | 2745 | 96.83 | 96.83 | 1873 | 95.39 | 96.10 | **83.59** | **89.70** |
| | Left (SC-3) | 399 | 91.77 | 89.47 | 2753 | 96.72 | 96.44 | 1869 | 94.86 | 95.77 | 83.13 | 89.29 |
| Czech | No Root (SC-1) | 382 | 80.51 | 74.61 | 2561 | 92.21 | 92.03 | 2057 | 90.14 | 91.59 | 68.30 | 81.14 |
| | Right (SC-2) | 380 | 81.07 | 75.53 | 2562 | 92.33 | 92.12 | 2058 | 90.19 | 91.59 | **73.96** | **81.16** |
| | Left (SC-3) | 406 | 83.33 | 72.66 | 2587 | 92.72 | 91.61 | 2007 | 88.28 | 91.93 | 72.98 | 79.96 |
| German | No Root (SC-1) | 397 | 94.68 | 85.14 | 2607 | 95.84 | 92.87 | 2004 | 90.49 | 95.96 | 83.85 | 86.64 |
| | Right (SC-2) | 397 | 94.68 | 85.14 | 2605 | 95.88 | 92.98 | 2006 | 90.64 | 96.01 | **83.93** | **86.72** |
| | Left (SC-3) | 354 | 89.64 | 90.40 | 2625 | 96.08 | 92.46 | 2029 | 90.64 | 94.92 | 83.29 | 86.08 |
| Slovene | No Root (SC-1) | 457 | 73.98 | 63.46 | 2255 | 88.07 | 91.35 | 2292 | 89.71 | 88.96 | 64.25 | 79.56 |
| | Right (SC-2) | 459 | 75.00 | 64.05 | 2250 | 88.03 | 91.51 | 2295 | 89.88 | 89.02 | **69.98** | **79.62** |
| | Left (SC-3) | 435 | 71.94 | 64.83 | 2300 | 88.11 | 89.61 | 2269 | 87.86 | 88.01 | 67.73 | 77.84 |
| Dutch | No Root (SC-1) | 510 | 65.56 | 66.08 | 2339 | 87.11 | 87.00 | 2149 | 87.34 | 87.30 | **71.09** | **74.51** |
| | Right (SC-2) | 507 | 65.76 | 66.67 | 2341 | 87.20 | 87.01 | 2150 | 87.34 | 87.26 | 71.05 | **74.51** |
| | Left (SC-3) | 641 | 72.18 | 57.88 | 2275 | 85.10 | 87.38 | 2082 | 86.73 | 89.48 | 70.81 | 74.41 |
| Chinese | No Root (SC-1) | 915 | 93.63 | 88.42 | 1134 | 88.99 | 91.98 | 2921 | 95.43 | 95.86 | 85.13 | 89.68 |
| | Right (SC-2) | 915 | 93.63 | 88.42 | 1134 | 88.99 | 91.98 | 2921 | 95.43 | 95.86 | **85.15** | **89.70** |
| | Left (SC-3) | 890 | 92.25 | 89.55 | 1160 | 88.91 | 89.83 | 2920 | 95.23 | 95.58 | 84.59 | 89.09 |
| Japanese | No Root (SC-1) | 1020 | 92.74 | 85.20 | 412 | 98.32 | 99.51 | 3571 | 95.92 | 98.01 | **89.85** | 92.10 |
| | Right (SC-2) | 1030 | 92.96 | 84.56 | 412 | 98.32 | 99.51 | 3561 | 95.70 | 98.06 | 89.77 | **92.12** |
| | Left (SC-3) | 926 | 88.15 | 89.20 | 418 | 98.56 | 98.33 | 3659 | 97.18 | 96.91 | 88.79 | 91.27 |
| Turkish | No Root (SC-1) | 645 | 90.29 | 92.25 | 254 | 82.99 | 94.09 | 4122 | 99.04 | 97.89 | **56.66** | **72.18** |
| | Right (SC-2) | 646 | 90.14 | 91.95 | 252 | 82.29 | 94.05 | 4123 | 99.04 | 97.87 | 56.64 | 72.16 |
| | Left (SC-3) | 629 | 88.77 | 93.00 | 245 | 79.51 | 93.47 | 4147 | 99.34 | 97.59 | 56.48 | 71.86 |

Table 4.3: Nivre arc-eager results for each of the three scenarios, showing root attachments, left attachments, right attachments and labeled (and unlabeled) attachment scores.

bic, Danish, Bulgarian, Spanish). This fact makes the parser suffers in the recall values. However, SC-1 and SC-2 get numbers that are closer to the proportion contained in the test corpus. This under production in SC-3 is normally affecting the attachments to the root. As a consequence, we observe an over-production of root attachments, probably most of them to the artificial (left) root node, we believe that the reason is basically that the artificial root node is considered as a node several times and the parser just tries to incorrectly attach nodes to it.

### Nivre arc-standard

In order to show that the results obtained with Nivre arc-eager were not a matter of luck, we decided to carry out the same experiment by running the parser with the Nivre arc-standard algorithm. We raise a new hypothesis: we do not expect the same improvements due to the arc-standard behavior when left attachments can be generated, however, we could expect some improvements in some cases by locating the root at the end of the sentence.

We again consider the same three scenarios. The results of the experiment for precision and recall of root attachments, left attachments, right attachments, LAS and UAS are shown in Table 4.4.

| | | Root Attachments | | | Left Attachments | | | Right Attachments | | | LAS and UAS | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Corpora | Root-Pos | # | Recall | Precision | # | Recall | Precision | # | Recall | Precision | LAS | UAS |
| Arabic | No Root (SC-1) | 299 | 81.69 | 80.60 | 4221 | 96.51 | 95.05 | 470 | 64.50 | 73.83 | 60.48 | 77.29 |
| | Right (SC-2) | 301 | 82.71 | 81.06 | 4216 | 96.46 | 95.11 | 473 | 64.50 | 73.36 | **65.29** | **77.31** |
| | Left (SC-3) | 302 | 83.73 | 81.79 | 4213 | 96.49 | 95.21 | 475 | 64.13 | 72.63 | 64.93 | 77.09 |
| Danish | No Root (SC-1) | 322 | 92.88 | 93.17 | 3734 | 98.14 | 97.43 | 954 | 90.61 | 93.08 | 81.64 | **87.86** |
| | Right (SC-2) | 325 | 92.88 | 92.31 | 3731 | 98.06 | 97.43 | 954 | 90.61 | 93.08 | 81.52 | 87.74 |
| | Left (SC-3) | 322 | 92.88 | 93.17 | 3736 | 98.17 | 97.40 | 952 | 90.51 | 93.17 | 81.66 | **87.86** |
| Bulgarian | No Root (SC-1) | 398 | 91.96 | 91.96 | 3198 | 98.00 | 97.94 | 1417 | 95.42 | 95.55 | 85.06 | **90.33** |
| | Right (SC-2) | 398 | 91.96 | 91.96 | 3198 | 98.00 | 97.94 | 1417 | 95.42 | 95.55 | **85.16** | **90.33** |
| | Left (SC-3) | 398 | 91.71 | 91.71 | 3201 | 98.03 | 97.88 | 1414 | 95.28 | 95.62 | 85.12 | **90.33** |
| Spanish | No Root (SC-1) | 197 | 81.73 | 81.73 | 3084 | 96.57 | 94.81 | 1710 | 91.17 | 94.15 | **77.88** | **81.69** |
| | Right (SC-2) | 196 | 80.71 | 81.12 | 3084 | 96.50 | 94.75 | 1711 | 91.11 | 94.04 | 77.72 | 81.55 |
| | Left (SC-3) | 197 | 81.22 | 81.22 | 3079 | 96.50 | 94.90 | 1715 | 91.34 | 94.05 | 77.64 | 81.51 |
| Portuguese | No Root (SC-1) | 288 | 90.28 | 90.28 | 3043 | 98.50 | 97.31 | 1678 | 95.51 | 97.62 | 78.32 | 87.78 |
| | Right (SC-2) | 288 | 90.62 | 90.62 | 3041 | 98.47 | 97.34 | 1680 | 95.57 | 97.56 | **83.47** | **87.80** |
| | Left (SC-3) | 288 | 90.62 | 90.62 | 3042 | 98.47 | 97.30 | 1679 | 95.51 | 97.56 | 83.45 | 87.82 |
| Swedish | No Root (SC-1) | 391 | 92.03 | 91.56 | 2750 | 97.19 | 97.02 | 1880 | 95.71 | 96.06 | **82.65** | **89.42** |
| | Right (SC-2) | 391 | 91.77 | 91.30 | 2750 | 97.16 | 96.98 | 1880 | 95.71 | 96.06 | 82.65 | 89.36 |
| | Left (SC-3) | 389 | 91.77 | 91.77 | 2752 | 97.16 | 96.91 | 1880 | 95.71 | 96.06 | 82.53 | 89.38 |
| Czech | No Root (SC-1) | 424 | 87.85 | 73.35 | 2540 | 92.72 | 93.31 | 2036 | 90.43 | 92.83 | 68.36 | 81.96 |
| | Right (SC-2) | 421 | 87.01 | 73.16 | 2540 | 92.68 | 93.27 | 2039 | 90.48 | 92.74 | 74.28 | 81.78 |
| | Left (SC-3) | 344 | 86.44 | 88.95 | 2617 | 94.68 | 92.47 | 2039 | 90.53 | 92.79 | **74.88** | **82.52** |
| German | No Root (SC-1) | 357 | 93.84 | 93.84 | 2607 | 96.20 | 93.21 | 2044 | 91.67 | 95.30 | 84.31 | 87.22 |
| | Right (SC-2) | 357 | 93.84 | 93.84 | 2607 | 96.16 | 93.17 | 2044 | 91.62 | 95.25 | 84.35 | 87.22 |
| | Left (SC-3) | 357 | 93.84 | 93.84 | 2598 | 96.00 | 93.84 | 2053 | 91.86 | 95.08 | **84.37** | **87.24** |
| Slovene | No Root (SC-1) | 438 | 75.26 | 67.35 | 2309 | 89.14 | 90.30 | 2257 | 88.87 | 89.50 | 63.67 | 79.28 |
| | Right (SC-2) | 436 | 75.26 | 67.66 | 2326 | 89.53 | 90.03 | 2242 | 88.47 | 89.70 | **69.42** | 79.28 |
| | Left (SC-3) | 366 | 73.47 | 78.69 | 2383 | 90.21 | 88.54 | 2255 | 88.96 | 89.67 | 69.40 | **79.42** |
| Dutch | No Root (SC-1) | 514 | 69.84 | 72.53 | 2366 | 87.33 | 86.22 | 2137 | 87.52 | 87.97 | 70.67 | 74.43 |
| | Right (SC-2) | 496 | 69.84 | 72.38 | 2366 | 87.33 | 86.22 | 2136 | 87.48 | 87.97 | 70.65 | 74.45 |
| | Left (SC-3) | 402 | 64.20 | 82.09 | 2453 | 90.37 | 86.06 | 2143 | 87.57 | 87.77 | **71.07** | **75.23** |
| Chinese | No Root (SC-1) | 864 | 93.06 | 93.06 | 1157 | 90.10 | 91.27 | 2949 | 96.56 | 96.07 | **85.25** | 90.08 |
| | Right (SC-2) | 864 | 92.82 | 92.82 | 1158 | 90.10 | 91.19 | 2948 | 96.52 | 96.07 | 85.17 | 90.00 |
| | Left (SC-3) | 864 | 92.82 | 92.82 | 1156 | 90.10 | 91.35 | 2950 | 96.59 | 96.07 | 85.23 | **90.10** |
| Japanese | No Root (SC-1) | 1015 | 92.53 | 85.42 | 412 | 98.08 | 99.27 | 3576 | 96.03 | 97.99 | **90.15** | **92.30** |
| | Right (SC-2) | 1014 | 92.64 | 85.60 | 412 | 98.08 | 99.27 | 3577 | 96.08 | 98.02 | 90.01 | 92.28 |
| | Left (SC-3) | 905 | 87.83 | 90.94 | 424 | 98.56 | 96.93 | 3674 | 97.70 | 97.03 | 89.13 | 91.57 |
| Turkish | No Root (SC-1) | 648 | 90.59 | 92.13 | 256 | 82.29 | 92.58 | 4117 | 99.02 | 97.98 | **57.00** | 72.06 |
| | Right (SC-2) | 648 | 90.59 | 92.13 | 255 | 82.29 | 92.94 | 4118 | 99.04 | 97.98 | 56.88 | 72.10 |
| | Left (SC-3) | 623 | 89.68 | 94.86 | 272 | 83.33 | 88.24 | 4126 | 99.21 | 97.96 | 56.80 | **72.12** |

Table 4.4: Nivre arc-standard results for each of the three scenarios, showing root attachments, left attachments, right attachments and labeled (and unlabeled) attachment scores.

The main conclusion is basically that our hypothesis is held, the arc-eager

transitions produced several spurious incorrect attachments when the root is to the left. This is not always happening with the arc-standard transitions, because it is more conservative when it generates left attachments. In fact, as we may observe in the experimental results, the three models are always very close, there are very slight differences between them and we even observe the same results in several cases.

As expected, for the Nivre arc-standard parsing algorithm the observed differences are small between one scenario to another. However, in SC-3 the results are normally the worst. We believe that the reason is that the arc-standard transitions may not produce the same spurious arcs as the arc-eager transitions, but still the artificial root node "is in play", and produces a few mistakes when the parser generates these left arcs to the node. And this fact is not happening in SC-1 and SC-2. It is more evidenced in the languages with a high proportion of left attachments (which are the ones shown first in the Table).

However, in some corpora (Czech, Slovene, Dutch and Japanese) we see that the number of root nodes of the scenario SC-3 drops substantially, which produces a drop in recall but an improvement in the precision (for root attachments). In these four cases, we see that in SC-3 the number of left attachments grows getting the attachments to the left root node. It is hard to explain why the parser behaves like this with arc-standard transitions, but we believe that the left artificial root node is (again) "in play" and therefore the parser just produces these attachments, while in SC-1 and SC-2 this is indeed not happening. Moreover, the differences in these four languages, are quite significant (normally 100 root nodes less). However in the rest of the corpora (all the other 9), the number of root attachments is the same, or just slightly different.

Comparing SC-1 to SC-2, we can show similar conclusions as the ones that we show with Nivre arc-eager, we believe that putting the root node at the end, we are basically producing the same behavior as not having root (or even improving the parsing performance, because we include the labeling and more information) the last step of the parser of generating root attachments. But, as stated, in a more informative way due to the root labels.

### 4.2.3 MSTParser Experiments

Finally, in order to see whether the root position is just a matter of study in transition-based dependency parsing, we carried out the same experiments with MSTParser (McDonald, Lerman, and Pereira, 2006), a graph-based parser. This kind of parser (also called Arc-Factored Spanning Tree Parsing) instead of scoring individual parsing actions, it scores all possible dependency arcs in the sentence and then uses exact inference to extract

the highest-scoring complete dependency tree under an arc-factored model,[5] where the score of each tree is the sum of the scores of its component arcs. Since the parsing algorithm does not impose any ordering at all on different attachments, we would expect even less impact from the placement of the dummy root node than for the deterministic arc-standard parser.

We modified the source code of MSTParser, forcing the parser to give no weight to the original root transitions, in this way we have exactly the same scenarios as we have in Section 4.2.2 with Nivre arc-eager and Nivre arc-standard. In this way in MaltParser and MSTParser, we attach to the artificial root in order to satisfy the tree constraint but without scoring the arcs going out of the artificial root, and in both cases it is possible to have more than one arc going out from the artificial root.

In this case we could expect the same outcomes from all the scenarios, or basically, just noise between one scenario to another.

The results are shown in Table 4.5. As expected, we can observe that the differences between one scenario to another with MSTParser are more or less random. In a graph-based approach this should be the case, because the root attachments do not interfere (at least, too much) as happened in a transition-based approach.

### 4.2.4 Conclusions

We believe there may be two main conclusions to extract.

- For certain parsing models, the existence and placement of the dummy root node is in fact a parameter worth tuning for best performance. As shown in the experiments explained in Section 4.2, for the deterministic Nivre arc-eager parser, we can obtain higher parsing accuracy by locating the dummy root node at the end of the sentence (or omitting it completely) instead of placing it at the beginning in the sentence, as is currently the norm in data-driven dependency parsing.

- The dummy root node may be an underestimated source of variation and a variable that needs to be controlled for in experimental evaluations. The current practice of consistently placing the root node at the beginning of the sentence is one way of ensuring comparability of results, but given the arbitrariness of this decision together with our experimental results, it may be worth exploring other representations as well.

[5] See Section 2.2

| | | Root Attachments | | | Left Attachments | | | Right Attachments | | | LAS and UAS | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Corpora | Root-Pos | # | Recall | Precision | # | Recall | Precision | # | Recall | Precision | LAS | UAS |
| Arabic | No Root (SC-1) | 296 | 84.07 | 83.78 | 4265 | 97.45 | 94.98 | 429 | 63.20 | 79.25 | **66.73** | **78.96** |
| | Right (SC-2) | 263 | 78.31 | 87.83 | 4284 | 97.64 | 94.75 | 443 | 64.13 | 77.88 | 66.41 | 78.32 |
| | Left (SC-3) | 272 | 83.39 | 90.44 | 4281 | 97.71 | 94.88 | 437 | 63.20 | 77.80 | 66.55 | 78.68 |
| Danish | No Root (SC-1) | 327 | 92.57 | 91.44 | 3713 | 98.17 | 91.44 | 970 | 98.17 | 98.01 | 83.39 | 89.46 |
| | Right (SC-2) | 323 | 92.26 | 92.26 | 3704 | 98.06 | 98.14 | 983 | 93.27 | 92.98 | 83.43 | 89.42 |
| | Left (SC-3) | 324 | 94.74 | 94.44 | 3717 | 98.30 | 98.04 | 969 | 92.65 | 93.70 | **83.97** | **89.84** |
| Bulgarian | No Root (SC-1) | 398 | 98.24 | 98.24 | 3211 | 98.75 | 98.29 | 1404 | 96.34 | 97.36 | 86.30 | **91.64** |
| | Right (SC-2) | 398 | 97.24 | 97.24 | 3208 | 98.56 | 98.19 | 1407 | 96.41 | 97.23 | 86.14 | 91.28 |
| | Left (SC-3) | 398 | 97.49 | 97.49 | 3198 | 98.44 | 98.37 | 1417 | 96.62 | 96.75 | **86.32** | 91.28 |
| Spanish | No Root (SC-1) | 200 | 79.70 | 78.50 | 3036 | 96.00 | 95.75 | 1755 | 93.15 | 93.73 | 79.40 | **83.57** |
| | Right (SC-2) | 200 | 84.77 | 83.50 | 3043 | 96.27 | 95.79 | 1748 | 93.32 | 94.28 | **79.48** | 83.53 |
| | Left (SC-3) | 196 | 83.76 | 84.18 | 3043 | 96.30 | 95.83 | 1752 | 93.26 | 94.01 | 79.20 | 83.41 |
| Portuguese | No Root (SC-1) | 288 | 89.58 | 89.58 | 2997 | 98.00 | 98.30 | 1724 | 97.43 | 96.93 | 84.87 | 89.74 |
| | Right (SC-2) | 289 | 90.62 | 90.31 | 2996 | 98.20 | 98.53 | 1724 | 97.61 | 97.10 | 84.89 | 89.26 |
| | Left (SC-3) | 288 | 91.67 | 91.67 | 2997 | 98.20 | 98.50 | 1724 | 97.61 | 97.10 | **85.19** | **90.26** |
| Swedish | No Root (SC-1) | 388 | 89.97 | 90.21 | 2735 | 96.25 | 96.60 | 1898 | 95.34 | 94.78 | 81.36 | 88.29 |
| | Right (SC-2) | 389 | 92.03 | 92.03 | 2722 | 96.10 | 96.91 | 1910 | 95.87 | 94.71 | 81.66 | 88.35 |
| | Left (SC-3) | 388 | 91.52 | 91.75 | 2723 | 96.25 | 97.03 | 1910 | 96.03 | 94.87 | **81.76** | **88.59** |
| Czech | No Root (SC-1) | 360 | 82.20 | 80.83 | 2592 | 94.64 | 93.33 | 2048 | 91.63 | 93.51 | 76.70 | 85.98 |
| | Right (SC-2) | 358 | 89.55 | 89.55 | 2592 | 94.99 | 93.67 | 2050 | 91.96 | 93.76 | **77.68** | **86.70** |
| | Left (SC-3) | 358 | 85.88 | 84.92 | 2608 | 96.31 | 93.40 | 2034 | 91.58 | 94.10 | 77.04 | 86.34 |
| German | No Root (SC-3) | 357 | 97.76 | 97.76 | 2569 | 96.63 | 95.02 | 2082 | 93.93 | 95.87 | 85.64 | 89.54 |
| | Right (SC-2) | 357 | 97.76 | 97.76 | 2564 | 96.44 | 95.01 | 2087 | 93.93 | 95.64 | 85.34 | 89.50 |
| | Left (SC-3) | 357 | 97.48 | 97.48 | 2563 | 96.52 | 95.12 | 2088 | 94.07 | 95.74 | **85.74** | **89.66** |
| Slovene | No Root (SC-1) | 408 | 79.08 | 75.98 | 2360 | 91.83 | 91.02 | 2236 | 90.37 | 91.86 | 71.44 | 82.47 |
| | Right (SC-2) | 380 | 76.79 | 79.21 | 2332 | 91.19 | 91.47 | 2292 | 91.60 | 90.84 | 71.64 | 82.33 |
| | Left (SC-3) | 392 | 79.34 | 79.34 | 2351 | 91.88 | 91.41 | 2261 | 90.98 | 91.46 | **71.72** | **82.67** |
| Dutch | No Root (SC-1) | 513 | 79.77 | 79.92 | 2347 | 90.97 | 90.54 | 2347 | 90.46 | 90.88 | **79.05** | **83.49** |
| | Right (SC-2) | 453 | 75.10 | 85.21 | 2386 | 91.61 | 89.69 | 2159 | 90.78 | 90.32 | 78.25 | 82.95 |
| | Left (SC-3) | 457 | 74.32 | 83.59 | 2394 | 92.17 | 89.93 | 2147 | 90.88 | 90.92 | 78.91 | 83.43 |
| Chinese | No Root (SC-1) | 864 | 94.33 | 94.33 | 1193 | 92.15 | 90.53 | 2913 | 96.05 | 96.74 | **86.88** | **90.82** |
| | Right (SC-2) | 863 | 93.87 | 93.97 | 1185 | 91.30 | 90.30 | 2922 | 96.01 | 96.41 | 86.36 | 90.52 |
| | Left (SC-3) | 864 | 94.33 | 94.33 | 1202 | 92.15 | 89.85 | 2904 | 95.74 | 96.73 | 86.54 | 90.68 |
| Japanese | No Root (SC-1) | 989 | 93.38 | 88.47 | 412 | 98.32 | 99.51 | 3602 | 96.82 | 98.08 | 90.45 | 93.02 |
| | Right (SC-2) | 953 | 92.21 | 90.66 | 409 | 97.84 | 99.76 | 3641 | 97.56 | 97.78 | 90.47 | 93.06 |
| | Left (SC-3) | 950 | 92.85 | 91.58 | 413 | 98.32 | 99.27 | 3640 | 97.75 | 97.99 | **90.83** | **93.36** |
| Turkish | No Root (SC-1) | 709 | 93.47 | 86.88 | 274 | 86.46 | 90.88 | 4038 | 97.69 | 98.56 | 58.49 | 74.55 |
| | Right (SC-2) | 649 | 93.02 | 94.45 | 273 | 85.42 | 90.11 | 4099 | 99.04 | 98.44 | **58.89** | **74.83** |
| | Left (SC-3) | 647 | 92.56 | 94.28 | 278 | 86.81 | 89.83 | 4096 | 98.94 | 98.41 | 58.59 | 74.59 |

Table 4.5: MSTParser results for each of the three scenarios, showing root attachments, left attachments, right attachments and labeled (and unlabeled) attachment scores.

## 4.3 Chapter Summary

During the present Chapter, we have shown a feasibility study and a succesful method to enhance the accuracy of MaltParser models.

About the parsing combination feasibility study proposed in Section 4.1, we believe that addressing the problem by solving small issues inside the parsers would make it beneficial in order to have a better parsing accuracy. Therefore, this work should be understood as a first attempt to solve these issues.

Moreover, we certainly believe that the work in which we studied the root position during parsing and training time, shown in Section 4.2, may change how the researchers think, principally the ones interested in transition-based

dependency parsing. We have provided consistent results in which we show that the root position is indeed relevant and it is something that developers must take into account in future parsing algorithms. We have shown that when a transition-based parser is greedy, generating left attachments there may be errors related to the root node when this one is located to the left of the sentences.

# Chapter 5

# Optimizing Dependency Analysis

*The value of experience is not in seeing much,*
*but in seeing wisely.*
*William Osler*

The development of accurate parsers for new languages may require careful optimization, a task that is often non-trivial especially for application developers that may lack the competence, the motivation adn the time needed to perform extensive parsing experiments. We can observe this fact even in this thesis, in the search of new feature models during Section 4.1, in the feasibility study about a parsing combination problem. This is why we wanted to put effort into this topic in order to come up with solutions that speed up this problem, and in a way provide even better results that make use of all the annotation provided in the treebanks.

As an illustration of the importance of optimization, Hall et al. (2007) report differences of over 3 percent absolute in labeled attachment score between the baseline version of MaltParser and the manually optimized system for some languages in the CoNLL 2007 shared task on dependency parsing. It is worth noting that using our methods, presented in this Chapter, these differences are greater than those typically reported when comparing different parsers on the same data sets, as we may see during the following Sections. Note that a manual optimization requires time, and a very deep knowledge about the task and the tool that is going to be used.

In this Chapter, we demonstrate that an automatic and optimal configuration of MaltParser is possible. Just running the system with default settings when training a new parser is likely to result in suboptimal performance, with respect to parsing accuracy as well as efficiency, but finding a good combination of all parameters can be a daunting task even for experi-

enced researchers.

Therefore, to facilitate MaltParser optimization, we propose a system, called MaltOptimizer,[1] that automates the search for optimal parameters based on an analysis of the training set and on optional input from the user at various points. Although the system is not guaranteed to find truly optimal settings, our experiments indicate that it invariably improves over the default settings and often approaches (or even surpasses) the results obtained through careful manual optimization.

## 5.1 What Do We Need to Optimize?

MaltParser, as we said in the previous chapters is a transition-based parser generator.[2] When optimizing MaltParser for a new language or domain, there are essentially three aspects of the system that need to be optimized:

1. Parsing algorithm
2. Feature model
3. Learning algorithm

We describe during the following Subsections each of these aspects in turn.

### 5.1.1 Parsing Algorithm

Selecting a parsing algorithm essentially means selecting a transition system together with certain constraints on search in that transition system. MaltParser implements four groups of transition-based parsing algorithms:

- Nivre's algorithms (Nivre, 2003; Nivre, 2008):
  - Nivre's arc-eager algorithm.
  - Nivre's arc-standard algorithm.
- Covington's algorithms (Covington, 2001; Nivre, 2008):
  - Covington's algorithm.
  - Covington's non-projective algorithm.
- Stack algorithms (Nivre, 2009; Nivre, Kuhlmann, and Hall, 2009):
  - Stack projective algorithm.
  - Stack non-projective algorithm.

[1] MaltOptimizer can be downloaded from http://nil.fdi.ucm.es/maltoptimizer

[2] See Section 2.1.1 in order to find a description of the transition-based parsing models.

- Multiplanar parsers (Gómez-Rodríguez and Nivre, 2010):
  - The Planar arc-eager parser.
  - The 2-Planar arc-eager parser.

MaltParser uses the Nivre arc-eager algorithm as default. Note also that both the Covington group and the Stack group contain algorithms that can handle non-projective dependency trees, and any projective algorithm can be combined with pseudo-projective parsing to recover non-projective dependencies in post-processing (Nivre and Nilsson, 2005).

The MaltParser algorithms are defined and described in Section 2.1.1, here we only show what we have to select in order to have the optimal output.

### 5.1.2 Feature Model

One of the advantages of the transition-based approach to dependency parsing is that it enables rich history-based feature models for predicting the next transition, and MaltParser provides an expressive specification language for defining feature models. Features are defined relative to tokens in the main data structures for a given parsing algorithm, which normally include at least a *stack* holding partially processed tokens and a *buffer* holding remaining input tokens. The actual feature values are normally linguistic attributes of one or more tokens based on some of the linguistic attributes of the CoNLL data format:[3]

1. FORM: Word form.
2. LEMMA: Lemma.
3. CPOSTAG: Coarse-grained part-of-speech tag.
4. POSTAG: Fine-grained part-of-speech tag.
5. FEATS: List of morphosyntactic features (e.g., case, number, tense, etc.)
6. DEPREL: Dependency relation to head.

The default model for a MaltParser parsing algorithm (which is what is used when the system is run with default settings) includes the following groups of features:

[3]See Section 1.1.2, in order to find a description the attributes of the CoNLL data format that can be features, you can also check http://nextens.uvt.nl/depparse-wiki/DataFormat to get a quick view of it.

1. A wide window of POSTAG features over the stack and buffer (typically of length 6).
2. A narrower window of FORM features over the stack and buffer (typically of length 3).
3. A small set of DEPREL features over dependents (and heads) of the most central tokens on the stack and in the buffer (typically of size 4).
4. A small set of combinations of the above features, in particular POSTAG n-grams and pairs of POSTAG and FORM features.

As we explain in Section 5.2.3, optimizing the feature model implies both tuning the size of these feature groups and exploring additional features such as CPOSTAG, LEMMA and FEATS features.

### 5.1.3 Learning Algorithm

MaltParser makes use of two libraries for machine learning: LIBSVM (Chang and Lin, 2001) and LIBLINEAR (Fan et al., 2008). The LIBSVM package enables the use of support vector machines with kernels, which facilitates feature selection but has the drawback of being rather inefficient both during training and parsing. The LIBLINEAR package only supports plain linear classifiers, which makes training and parsing very fast but put higher demands on feature selection. Both packages contain a number of specific algorithms each with their own hyperparameters which is what we need to optimize. Note also that in MaltParser both LIBSVM and LIBLINEAR provide outcomes in the same range of accuracy, or as stated in (Prudhvi Kosaraju and Kukkadapu, 2010) or (Gómez-Rodríguez and Fernández-González, 2012), LIBLINEAR could even provide better results than LIBSVM, but for other languages LIBSVM is better. Nevertheless, LIBLINEAR is always faster.

For the development of MaltOptimizer, and in the rest of the experiments and assumptions of the present Chapter, we have restricted our attention to the LIBLINEAR package in the interest of efficiency. MaltParser uses LIBSVM as default learning library, however, it is very likely that this fact will change in future versions of MaltParser.

MaltOptimizer tunes the LIBLINEAR cost parameter $C$, which as stated by Fan et al. (2008), it is the only parameter for linear classification. The $C$ parameter enforces the margin between the different classes and compensates that some points might be misclassified. A high $C$ value could produce overfitting, providing a model with higher training error but lower test error, due to a harder margin (Cassel, 2009). We only explored the $C$ parameter for values between 0 and 1, because when it is greater than 1 it could be understood as a penalty parameter, and it is not interesting to optimize MaltParser models.

## 5.2 MaltParser Optimization: MaltOptimizer

In order to do all the things shown in Section 5.1, we developed MaltOptimizer. MaltOptimizer is a software tool written in Java that implements a stepwise optimization procedure for MaltParser based on the heuristics described in (Nivre and Hall, 2010). The system takes as input a training set, consisting of sentences annotated with dependency trees in the CoNLL-X data format. The optimization process has three different phases with optional input from the user after each phase:

1. Data validation, data analysis and initial optimization.

2. Parsing algorithm selection.

3. Feature selection and LIBLINEAR hyper-parameter optimization.

MaltOptimizer estimates the expected results by providing labeled attachment score results (LAS).[4] MaltOptimizer uses the evaluation measure (LAS, either with or without punctuation symbols) in order to select the optimal configuration possible through all the optimization process by for instance making the tests between one algorithm versus another one (Phase 2, Section 5.2.2), or by selecting one new feature or not selecting it (Phase 3, Section 5.2.3). Note that the user may select the evaluation measure used to estimate the results, in default settings it is LAS, but it can be changed to either UAS or LCM.

Note that the machine learning algorithm used throughout the process is the multiclass support vector machine of (Crammer et al., 2006) as implemented in LIBLINEAR (Fan et al., 2008). We describe in the following subsections each of the three phases in turn.

### 5.2.1 Phase 1: Data Analysis and Initial Optimization

During the Phase 1, MaltOptimizer makes an analyisis of the data set, suggests the best validation strategy and performs some initial optimizations.

#### Data Analysis

MaltOptimizers starts by validating that the data is correctly formatted, using the official validation script from the CoNLL-X shared task (*validateFormat.py*[5]). If the data is not valid according to the validation script, MaltOptimizer stops the process and informs the user to check it and fix

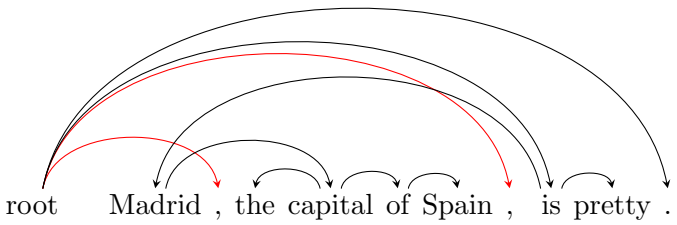
[4] In the default settings, it provides LAS including punctuation symbols, but it can be configured excluding punctuation symbols and excluding the labeling returning unlabeled attachment scores (UAS).

[5] The script can be downloaded from http://ilk.uvt.nl/conll/software.html

it, if possible. If the script returns warnings showing possible inconsistencies, the system proceeds with the data analysis but recommends the user to check the script output since the warnings may indicate problems in the annotation.

In the data analysis, MaltOptimizer gathers information about the following properties of the training set:

1. Number of words/sentences.
2. Percentage of non-projective arcs/trees.
3. Existence of "covered roots" (a root node (HEAD = 0) covered by an arc that is not connected to the root). The dependency tree, that shows the tree of the sentence *'Madrid, the capital of Spain, is pretty.'*, shown below has two covered roots: (i) The one between the root and the first comma is covered by the arc between *Madrid* and *capital* which is not directly connected to the root. (ii) The one between the root node and the second comma is covered by the arc between *is* and *Madrid* which is not directly connected to the root.




4. Frequency of labels used for tokens with HEAD = 0. Some data sets use different labels (different than *ROOT*), such as the Czech, Portuguese, Arabic or Slovene treebanks.
5. Existence of non-empty feature values in the LEMMA and FEATS columns.
6. Identity (or not) of feature values in the CPOSTAG and POSTAG columns.

This information is going to be used during all the following optimization steps.

### Validation Strategy

Based on the size of the data set, MaltOptimizer recommends the user to choose one of the validation methods during Phase 2 and 3:

1. Simple train-devtest split (80% for training, 20% for development testing). This validation method is recommended for large data sets with more than 90,000 wordforms, where cross-validation would be time consuming and a single devtest set is large enough to give reliable estimates.

2. 5-fold cross validation (20% in each fold). This method is recommended for smaller data sets, where a single evaluation estimate might be unreliable and the extra time needed to run cross-validation is tolerable.

In both cases, the system performs stratified sampling to ensure a similar distribution of data in all subsets. In either case, the user can override the system recommendation.[6]

### Initial Optimization

Finally, MaltOptimizer makes some initial tests with the default parsing algorithm, which, as stated above, is Nivre arc-eager in default settings:

- If there are covered roots, MaltOptimizer tests the options of the flag `-pcr`, which can be *none* (default), *left*, *right* and *head*. MaltParser basically attaches the covered roots where it is stated in the flag, either right, left or to the root.[7] MaltOptimizer selects the option that provides the best outcome.

- If there is more than one label used for tokens with HEAD = 0, MaltOptimizer makes a single test for each in order to decide which one should be selected as main root label. This test is performed by modifying the `-grl` option provided in MaltParser. MaltOptimizer selects the root label that provides the best outcome.

When the data analysis and initial optimization tests are completed, MaltOptimizer creates a baseline option file and a log file to be used as the starting point for optimization. The user is given the opportunity to edit this option file before optimization continues and may also choose to stop the process and continue with manual optimization, for instance, modifying the default root label or the optimal way of handling covered roots. The Figure 5.1 shows an example option file and an example log file after Phase 1.

[6] See Appendix C to find experiments with either train dev-test split and cross-validation.

[7] See www.maltparser.org/optiondesc.html in order to find a complete description of each option.

```
phase1_logfile.txt

Training set path:slovene_sdt_train.conll
Size (tokens):28750
Size (sentences):1534
Non projective:22.164276401564535
Dangling Punctuation:27
LAS:66.15
Default:65.63
NumRootLabels:11
JavaHeap:-Xmx2048m
MaxTokens:2147483647
CposEqPos:false
LemmaBlank:false
FeatsBlank:false

---------------------

phase1_optfile.txt

1. root_label(-grl):Pred
2. covered_root(-pcr):right
```

Figure 5.1: Phase 1 options file and log file. The option file ($phase1\_optfile.txt$) shows the optimal outcomes of the initial optimization process, the user may edit it. The log file ($phase1\_logfile.txt$) shows the main characteristics of the data set plus some hints that are going to be used during the rest of the optimization process.

### 5.2.2 Phase 2: Parsing Algorithm Selection

In the second phase, MaltOptimizer explores the parsing algorithms implemented in MaltParser, based on the results of the data analysis and after that it tunes the specific options of each parsing algorithm when they exists.

#### Parsing Algorithm Selection

MaltOptimizer explores the parsing algorithms implemented in MaltParser:

- If there are no non-projective arcs/trees in the training set, then only projective algorithms are explored:
  - The arc-eager and arc-standard versions of Nivre's algorithm (Nivre, 2003; Nivre, 2004).
  - Covington's projective parsing algorithm (Covington, 2001; Nivre, 2008).
  - The projective Stack algorithm (Nivre, 2009).
  - The Planar arc-eager parser (Gómez-Rodríguez and Nivre, 2010).
- If the training set contains more than 15% of sentences with at least one non-projective arc then MaltOptimizer instead tests the following algorithms:

  - Covington's non-projective algorithm (Covington, 2001; Nivre, 2008).
  - The non-projective Stack algorithms (Nivre, 2009; Nivre, Kuhlmann, and Hall, 2009)
  - The 2-Planar arc-eager parser (Gómez-Rodríguez and Nivre, 2010).
  - Projective algorithms in combination with pseudo-projective parsing (Nivre and Nilsson, 2005).

- Otherwise both groups of algorithms are explored and MaltOptimizer finally selects the best one.

In order to reduce the number of tests needed, we came up with two different decision trees based on previous experience (Nivre and Hall, 2010). The first one, shown in Figure 5.2, tests only projective algorithms in such a way that the maximum number of tests is 4, and the procedure avoids unnecessary tests such as testing the Nivre arc-standard algorithm when the Nivre arc-eager algorithm provides better results than the projective Stack algorithm due to parsing order constraints. The Nivre arc-standard algorithm uses the same parsing order as the projective Stack algorithm and this is why we can remove unnecesary tests. Moreover, Planar arc-eager is similar to Nivre arc-eager considering parsing order and this is why it is collocated in the same branch as Nivre arc-eager.

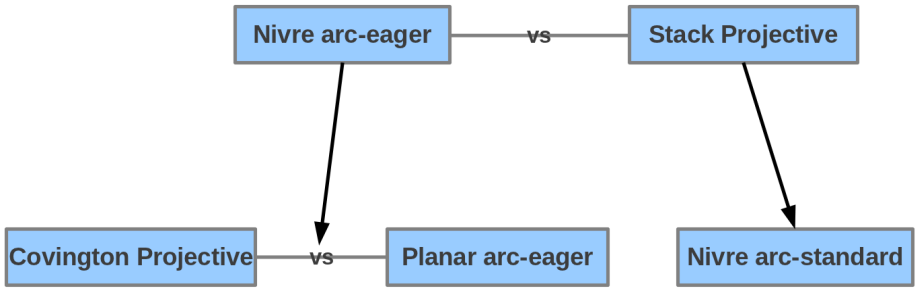



Figure 5.2: Decision tree for best projective algorithm. Each box of the decision tree means that MaltOptimizer tests this parsing algorithm. 'vs' means that MaltOptimizer decides between two parsing algorithms testing both and selecting the one with better results according to the selected evaluation measure (LAS or UAS). A single arrow means that MaltOptimizer only tests the algorithm that is collocated below if the predecessor provides the best results so far.

The second decision tree, shown in Figure 5.3, is for non-projective algorithms and results in a maximum of 6 tests using similar considerations. However, in this case we have a larger number of algorithms to consider.

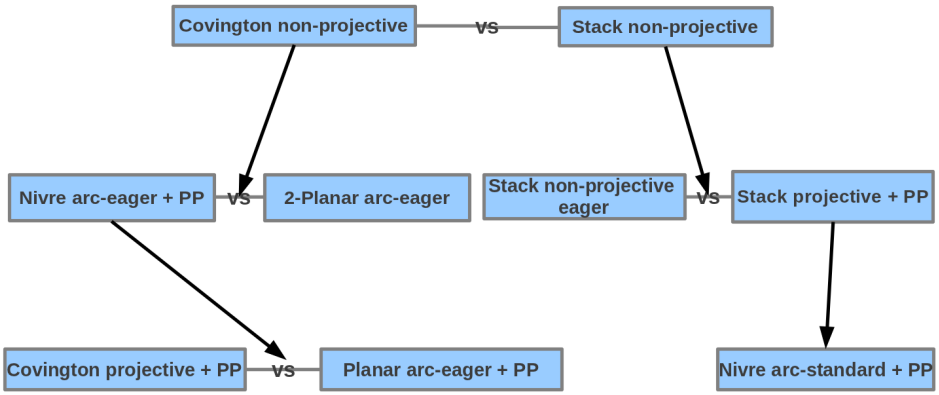



Figure 5.3: Decision tree for best non-projective algorithm (+PP for pseudo-projective parsing). Each box of the decision tree means that MaltOptimizer tests this parsing algorithm. 'vs' means that MaltOptimizer decides between two parsing algorithms testing both and selecting the one with a better outcome according to the selected evaluation measure (LAS or UAS). A single arrow means that MaltOptimizer only tests the algorithm that is collocated below if the predecessor provides the best results so far.

### Parameter Optimization

After traversing one or both of these decision trees with the default settings, MaltOptimizer tunes the parameters of the best performing algorithm and creates a new option file for the best configuration, according to the evaluation measure. We here show specifically the tests that MaltOptimizer performs for each algorithm or algorithm family when available:

- **Nivre's algorithms**: MaltOptimizer tests the root handling selected strategy, by performing tests with the boolean MaltParser options `allow_root` and `allow_reduce`.
  - `allow_root`=true (default) means that the parser uses the root node as another token during parsing. Otherwise, there is no root token during parsing.
  - `allow_reduce`=false (default) means that the reduce transition is not permitted if the node on the top of the stack is a root node. Otherwise, it is permitted and it will be attached to the root node in the last parsing step.
- **Covington's algorithms**: MaltOptimizer tests again the root handling strategy, it performs tests over the option `allow_root` (but in the Covington's group of options), and it tests the `allow_shift` option.

  - `allow_root`=true (default) means that the parser uses the root node as another token during parsing. Otherwise, there is no root token during parsing.
  - `allow_shift`=false (default) means that the transition *Shift* is not permitted. Otherwise, it is permitted.

- **2-Planar arc-eager**: MaltOptimizer tests the options `reduceonswitch` (boolean) and the planar root handling options `planar_root_handling`.
  - `reduceonswitch`=false (default) means that the parser does not reduce when it makes the *switch* operation. Otherwise, it reduces.
  - `planar_root_handling`=normal (default) means that root dependents are attached by RightArc transitions.
    `planar_root_handling`=relaxed, the root nodes not attached during parsing are attached with default label as in `allow_root`=false for Nivre's algorithm.

- **Pseudo-Projective parsing algorithm**: if MaltOptimizer decides that the best parsing algorithm is a projective algorithm run with pseudo-projective parsing (see +PP, in Figure 5.3), then it can test several options for the flag `pp`.
  - *baseline*: projectivizes input data.
  - *head*: projectivizes with head encoding for labels.
  - *path*: projectivizes with path encoding for labels.
  - *head+path*: projectivizes with head and path encoding for labels.

At the end of Phase 2, the user is again given the opportunity to edit the option file (or stop the process) before optimization continues. For instance, the user may change the parsing algorithm or a single option selected by MaltOptimizer as optimal. The Figure 5.4 shows an example option file and an example log file after Phase 2.

### 5.2.3 Phase 3: Feature Selection

In the third phase, MaltOptimizer tries to optimize the feature model given the parameters chosen so far (in particular the parsing algorithm). It first performs backward selection experiments to ensure that all features in the default model for the given parsing algorithm actually make a contribution. It then proceeds with forward selection experiments, trying potentially useful features one by one and in combination.

**Backward selection** could be presented in a formal fashion as follows:

Let $X = X_1, \ldots, X_n$ be the given set of features and let M($X$) be the evaluation metric for $X$ (in our case, either LAS or UAS).

```
phase2_logfile.txt

Training set path:slovene_sdt_train.conll
Size (tokens):28750
Size (sentences):1534
Non projective:22.164276401564535
Dangling Punctuation:27
LAS:68.34
Default:65.63
NumRootLabels:11
JavaHeap:-Xmx2048m
MaxTokens:2147483647
CposEqPos:false
LemmaBlank:false
FeatsBlank:false

---------------------

phase2_optfile.txt

1. root_label (-grl):Pred
2. covered_root (-pcr):right
3. parsing_algorithm (-a):stacklazy
```

Figure 5.4: Phase 2 options file and log file. The option file ($phase2\_optfile.txt$) shows the optimal outcomes of the initial optimization process, the user may edit it. The log file ($phase2\_logfile.txt$) shows the main characteristics of the data set plus some hints that are going to be used during the rest of the optimization process.

*While* $|X| < 1$
  $B = 0$
  $Best = NIL$
  *For each* $X_i \in X$
    *If* $M(X - X_i) > B$ *then*
      $B = M(X - X_i)$
      $Best = X_i$
  *If* $B < M(X)$ *then*
    *return* $X$
  *Else*
    $X = X - X_i$
*Return* $X$

Following this algorithm the number of possibilities grows exponentially, making exhaustive search impractical for even moderate sizes of $X$.

**Forward Selection**. Normally, start with no variables and add the possible features one by one, at each step adding the one that performs better in a significant way.

Forward selection could be presented in a formal fashion as follows:

Let X = $X_1, \ldots, X_n$ be the pool of potential features and let M(X) be the evaluation metric for $X$ (in our case, either LAS or UAS).

```
Let Y = { }
While Y ≠ X
    B = 0
    Best = NIL
    For each X_i ∈ X
      If M(Y U X_i) >B then
        B = M(Y U X_i)
        Best = X_i
    If B < M(X) then
      Return Y
    Else
      Y = Y U X_i
      X = X - X_i
Return Y
```

An exhaustive search for the best possible feature model is practically impossible, so our optimization strategy is based on heuristics derived from proven experience (Nivre and Hall, 2010). Therefore, the major steps of the forward and backward selection experiments are the following:

1. Tune the window of POSTAG (n-gram) features over the stack and buffer.
2. Tune the window of (lexical) FORM features over the stack and buffer.
3. Tune dependency tree features using DEPREL and POSTAG features.
4. Add predecessor and successor features for salient tokens using POSTAG and FORM features.
5. Add CPOSTAG, FEATS, and LEMMA features if available.
6. Add conjunctions of POSTAG and FORM features.

Our algorithm traverses all the 6 steps adding one feature at a time and keeping the feature set that provides the best result so far in a greedy fashion. We speed up the process by applying the following heuristics acquired from previous experience:[8]

1. If backward selection provides improvements for a specific window, we do not try forward selection for this window.
2. As soon as forward selection is unsuccessful for a specific window, we do not try further forward selection experiments for this window.

[8]Please, find Appendix C for experiments with automatic feature selection that emphasize these heuristics.

These six steps are slightly different depending on which algorithm is the best with default settings, because the MaltParser algorithms have different parsing orders and use different data structures, but the steps are roughly equivalent at a certain level of abstraction. Table 5.1 shows the differences between the data structures of each different family of algorithms implemented in MaltParser. The process of adding features between the stack and buffer are quite similar taking into account that the Stack algorithms add words to the stack in a lazy way, and LookAhead is the equivalent structure to the Input (or buffer) in the Nivre's family. Covington's family contains two different data structures that are equivalent to the Input and the Stack in Nivre's family. Stack algorithms also contain another data structure that is called Input,[9] Covington non-projective algorithm contains two extra and different data structures: (i) LeftContext and (ii) RightContext, and the 2-Planar parser contains an extra data structure: the InactiveStack.

| **Algorithm (or family)** | **Structures** | **Stack** | **Buffer** | **Top of the Stack** | **1st Position Buffer** | **2nd Position Buffer** |
|---|---|---|---|---|---|---|
| Nivre's family | Stack, Input | Stack | Input | Stack[0] | Input[0] | Input[1] |
| Stack family | Stack, Input LookAhead | Stack | LookAhead | Stack[1] | Stack[0] | LookAhead[0] |
| Covington projective | Left, Right | Left | Right | Left[0] | Right[0] | Right[1] |
| Covington non-projective | Left, Right, LeftContext RightContext | Left | Right | Left[0] | Right[0] | Right[1] |
| Planar | Stack, Input | Stack | Input | Stack[0] | Input[0] | Input[1] |
| 2-Planar | ActiveStack, Input InactiveStack | ActiveStack | Input | ActiveStack[0] | Input[0] | Input[1] |

Table 5.1: Table of equivalences between data structures and relevant positions in each family of algorithms.

After the feature selection experiments are completed, MaltOptimizer creates a new option file and a new feature specification file. The user is given the opportunity to edit both of these files (or stop the process) before optimization continues.

### LIBLINEAR Parameter Optimization

At the end of the third phase, MaltOptimizer tunes the $C$ parameter of the multiclass SVM (LIBLINEAR) using a simple grid search, starting in $C$=0.01, iterating ten times until $C$=1.0, selecting the optimal $C$ value, which is the one that shows a higher LAS (or UAS). After this optimization is completed, MaltOptimizer creates the final option file. The user may now continue to do further optimization manually. The Figure 5.5 shows an example option file and an example log file after Phase 3.

[9] But this *Input* is a different one from the Input in Nivre's family, the equivalent structure to the Input (in the Nivre's family) in the Stack family is called *LookAhead*. See Table 5.1

```
phase3_logfile.txt

Training set path:slovene_sdt_train.conll
Size (tokens):28750
Size (sentences):1534
Non projective:22.164276401564535
Dangling Punctuation:27
LAS:71.65
Default:65.63
NumRootLabels:11
JavaHeap:-Xmx2048m
MaxTokens:2147483647
CposEqPos:false
LemmaBlank:false
FeatsBlank:false

---------------------

phase3_optfile.txt

1. root_label (-grl):Pred
2. covered_root (-pcr):right
3. parsing_algorithm (-a):stacklazy
8. feature model (-F):addMergPOSTAGS0FORMStack1.xml
```

Figure 5.5: Phase 3 options file and log file. The option file (*phase2_optfile.txt*) shows the optimal outcomes of the initial optimization process. The numbering jumps from 3 to 8 due to the parsing algorithm selected, the slots 4-7 serve for the parsing algorithm options, but in this case is `stacklazy` (or Stack non-projective) and it does not have further options as we may see in Section 5.2.2. The log file (*phase2_logfile.txt*) shows the main characteristics of the data set plus some hints that have been used during the optimization process.

## 5.3 Experiments with MaltOptimizer

In order to assess the usefulness and validity of the optimization procedure, we have run all three phases of the optimization on all the data sets from the CoNLL 2006 and 2007 shared tasks on dependency parsing (Buchholz and Marsi, 2006; Nivre et al., 2007). We used 5-fold cross-validation for all training sets smaller than 90,000 words and a simple train-devtest split for the larger training sets, by using the suggestions that we provide in MaltOptimizer.[10]

Table 5.2 shows LAS with default settings and after each of the three optimization phases, as well as the difference between the final configuration and the default. The last two columns compare the accuracy obtained on the final test set (Test-MO) with the best score obtained with manual optimization of MaltParser (Test-MP) in the original shared tasks (Nivre et al., 2006a; Hall et al., 2007). Table 5.3 shows the algorithm selected by MaltOptimizer for each of the data sets after running the Phase 2.

The first thing to note is that the optimization improves parsing accuracy

[10]See, Appendix C to find a more detailed feature selection experiment emphasizing the use of K-fold cross validation.

| **CoNLL-X Shared Task** | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Language** | **Default** | **Phase 1** | **Phase 2** | **Phase 3** | **Diff** | **Test-MO** | **Test-MP** |
| Arabic* | 63.02 | 63.03 | 64.03 | 66.37 | 3.35 | 66.20 | **66.71** |
| Bulgarian | 83.19 | 83.19 | 84.00 | 86.03 | 2.84 | 86.44 | **87.41** |
| Chinese | 84.14 | 84.14 | 84.95 | 84.95 | 0.81 | 85.49 | **86.92** |
| Czech | 69.94 | 70.14 | 72.44 | 78.04 | 8.10 | **80.46** | 78.42 |
| Danish | 81.01 | 81.01 | 81.34 | 83.86 | 2.85 | 83.41 | **84.77** |
| Dutch | 74.77 | 74.77 | 78.02 | 82.63 | 7.86 | 77.23 | **78.59** |
| German | 82.36 | 82.36 | 83.56 | 85.91 | 3.55 | 85.24 | **85.82** |
| Japanese | 89.70 | 89.70 | 90.92 | 90.92 | 1.22 | 90.39 | **91.65** |
| Portuguese | 84.11 | 84.31 | 84.75 | 86.52 | 2.41 | 85.85 | **87.60** |
| Slovene* | 66.08 | 66.52 | 67.86 | 72.29 | 6.21 | **73.66** | 70.30 |
| Spanish* | 76.45 | 76.45 | 76.64 | 79.65 | 3.20 | 80.18 | **81.29** |
| Swedish | 83.34 | 83.34 | 83.50 | 84.09 | 0.75 | 83.81 | **84.58** |
| Turkish* | 57.79 | 57.79 | 58.33 | 67.11 | 9.32 | 64.85 | **65.68** |
| **CoNLL 2007 Shared Task** | | | | | | | |
| **Language** | **Default** | **Phase 1** | **Phase 2** | **Phase 3** | **Diff** | **Test-MO** | **Test-MP** |
| Arabic | 67.71 | 67.75 | 67.75 | 70.77 | 3.06 | 73.22 | **74.75** |
| Basque* | 67.69 | 67.83 | 68.29 | 75.05 | 7.36 | 72.19 | **74.99** |
| Catalan | 83.07 | 83.07 | 83.13 | 84.89 | 1.82 | 85.87 | **87.74** |
| Chinese | 84.04 | 84.04 | 85.03 | 86.21 | 2.17 | 82.58 | **83.51** |
| Czech | 70.25 | 70.51 | 72.49 | 77.71 | 7.46 | **78.03** | 77.22 |
| English | 83.84 | 83.84 | 85.34 | 86.61 | 2.77 | 85.17 | 85.81 |
| Greek* | 71.01 | 71.09 | 72.41 | 75.12 | 4.11 | **74.50** | 74.21 |
| Hungarian | 66.42 | 66.42 | 68.21 | 76.53 | 10.11 | 77.17 | **78.09** |
| Italian* | 79.07 | 79.07 | 79.45 | 81.53 | 2.46 | **82.79** | 82.48 |
| Turkish* | 67.45 | 68.38 | 70.67 | 76.91 | 9.46 | 78.93 | **79.24** |

Table 5.2: LAS per phase compared to default settings for all training sets from the CoNLL-X shared task and the CoNLL 2007 Shared task. Languages marked * have a training set smaller than 90,000 tokens and have been optimized using 5-fold cross-validation; the remaining languages have been optimized using a simple train-devtest split. The last two columns report LAS on the final test sets for the best model found by MaltOptimizer (Test-MO) and the best MaltParser model in the original shared tasks (Test-MP)

for all languages without exception, although the amount of improvement varies considerably from about 1 percentage point for Chinese, Japanese and Swedish to 7–10 points for Basque, Dutch, Czech, Hungarian and Turkish. Part of the explanation for these differences is the fact that some data sets include rich linguistic annotation, which makes it beneficial to enrich the feature model by adding new features based of morphosyntactic information.

This is also why, for most languages, the greatest improvement comes from feature selection in phase 3. However, we also see significant improvement from phase 2 for languages with a substantial amount of non-projective

| CoNLL-X Shared Task | |
|---|---|
| **Corpora** | **Algorithm** |
| Arabic | Stack non-projective |
| Bulgarian | Stack non-projective |
| Chinese | Covington projective |
| Czech | Stack non-projective |
| Danish | Stack non-projective |
| Dutch | Nivre arc-eager + PP |
| German | Covington non-projective |
| Japanese | Covington projective |
| Portuguese | Stack non-projective |
| Slovene | Stack non-projective |
| Spanish | Nivre arc-eager |
| Swedish | Nivre arc-eager + PP |
| Turkish | Covington non-projective |
| **CoNLL 2007 Shared Task** | |
| **Corpora** | **Algorithm** |
| Arabic | Nivre arc-eager + PP |
| Basque | Covington non-projective |
| Catalan | Nivre arc-eager |
| Chinese | Stack projective |
| Czech | Stack projective + PP |
| English | Nivre arc-standard + PP |
| Greek | Stack non-projective |
| Hungarian | Stack projective + PP |
| Italian | Nivre arc-eager + PP |
| Turkish | Covington non-projective |

Table 5.3: Algorithms selected by MaltOptimizer (after Phase 2) for each data set. PP means pseudo-projective parsing (Nivre and Nilsson, 2005).

dependencies, such as Czech, Dutch and Slovene, where the selection of parsing algorithm is quite important (see Table 5.3). In these cases, a non-projective algorithm is often selected and provides the best results in default settings. See Appendix D in which we show an experiment in which we interacted with MaltOptimizer, forcing it to run a feature selection experiment just with the Multiplanar (Planar and 2-Planar) parsers.

Turning to the final test results, we see that MaltOptimizer performs competitively compared to the manually optimized version of MaltParser, with an average difference of only 0.61 and a maximum (negative) difference of 1.87 (for Catalan 2007). Moreover, we see that in 5 cases out of 23, Malt-

Optimizer actually improves on the old results, and in a few cases with a quite substantial margin. In fact, in the 2006 shared task, MaltOptimizer would have finished third, beaten only by MSTParser (McDonald, Lerman, and Pereira, 2006) and MaltParser (Nivre et al., 2006a), which we believe that for an automated configuration system is a very interesting outcome. Even more, if we compare the time and knowledge needed in order to get results in the same range of accuracy.

The time needed to run the optimization depends primarily on the size of the training set and the validation method chosen. With a simple train-devtest split, it ranges from about half an hour for the smallest data set (Slovene) to about one day for the largest one (Czech 2006). With 5-fold cross-validation, it will basically take five times longer. It is worth noting that the time needed depends deeply on the computer in which we run the experiments, the estimates given were obtained with a Intel Xeon server, several cores, 2.8Ghz and a big amount of dynamic memory.

Note that the experiments also show that the LIBLINEAR $C$ parameter is constantly better with the default MaltParser value, which is 0.1, however, for reasons of completion it is still worth exploring it during Phase 3, because with smaller data sets that could be different and a modification could provide some improvements.

Finally, it is worth noting that we tried to alter the order between the different steps shown in Section 5.2.3, but we did not get any improvement nor any significant differences between the different feature sets and the different algorithms. The results were not the same but we did not come up with anything conclusive, because a specific order for a treebank was useful and better, but it was not the same for a different treebank.

## 5.4 Conclusions

MaltOptimizer is an optimization tool for MaltParser that can support developers in adapting the system to new languages. We have demonstrated that by using MaltOptimizer it is possible to get substantial improvements over the default settings, thereby allowing non-experts in dependency parsing to achieve high (if not optimal) accuracy, which is usually not possible when using the system "out of the box".

In addition to application developers, MaltOptimizer should also be useful for people doing parsing research and who want to use MaltParser as a point of comparison for their own systems, since the results obtained with default settings can be highly misleading. Moreover, in MaltOptimizer, it is also possible to influence the optimization process at various points, which should make the system potentially useful also for expert users in order to speed up the optimization process or even provide better results than the ones obtained by careful and expert manual optimization.

We believe it is possible to extend this kind of technique to different dependency parsers that use a rich feature set specification. In the case of transition-based parsers it could be handled simply by transforming the produced feature set for the new feature specification language. In other kind of parsers, for instance, maximum spanning tree parsers, the features are simple because we do not have to worry about the data structures that are present in transition-based parsing but we could consider the same kind of features: part-of-speech, morphology, etc. Moreover, it is worth remarking that our methods can be applied to all MaltParser algorithms, we therefore believe that they could provide a universal way of tuning the features of transition-based parsers.

## 5.5 Chapter Summary

In this Chapter we have presented MaltOptimizer, which is an optimization tool for MaltParser. It is primarily aimed at application developers who wish to adapt the system to a new language or domain and who do not have expert knowledge about transition-based dependency parsing. Another potential user group consists of researchers who want to perform comparative parser evaluation, where MaltParser is often used as a baseline system and where the use of suboptimal parameter settings may undermine the validity of the evaluation. Finally, we believe the system can be useful also for expert users of MaltParser as a way of speeding up the optimization procedure.

# Chapter 6

# Applying Dependency Analysis

*Let each man exercise the art he knows.*
*Aristophanes*

In this Chapter we present two applications in which we made use of the usefulness of dependency structures in order to solve some diverse NLP problems. In particular, we present work in the following applications:

- In Section 6.1 we show a system that simplifies sentences by developing a dependency tree pruning algorithm.
- In Section 6.2 we show a system that is able to infer the scope of negation cues.

The related work to the studies and experiments presented here are only important for this Chapter. This is why we here show specific related work to the problems presented in the following Sections.

## 6.1 Text Simplification for Spanish via Dependency Analysis

In this Section we investigate the task of text simplification for Spanish by making use of dependency structures. Our main motivation was the need of simplified texts in order to facilitate accessibility to information by people with cognitive disabilities. We were willing to show that dependency parsing can be very useful in order to achieve this goal. This study consists of a first step towards building Spanish text simplification systems helping to create easy-to-read texts.

There are a vast number of people that fall under the so called rudimentary and basic literacy levels. These people are only able to find explicit information in short texts or process simple sentences and make easy inferences. According to some studies (*http://www.facillectura.es*) which measure the literacy level of the population, in Spain, as an example, 30% of the population have difficulties understanding texts beyond a certain level of complexity.

Reading comprehension entails three elements:

- the reader who has to understand what he/she reads;
- the text that is to be comprehended and
- the activity in which comprehension is a part of, as mentioned in (Snow et al., 2002).

It has already been shown that long sentences, conjoined sentences, embedded clauses, passives, non-canonical word order, and use of low-frequency words, among other things, increase text complexity for language-impaired readers (Siddharthan, 2002), (Klebanov, Knight, and Marcu, 2004), (Devlin and Unthank, 2006), (Caseli et al., 2009). There are different initiatives that make available guidelines to make text easier to comprehend: the Plain Language (*http://www.plainlanguage.gov*) or "European Guidelines for the Production of Easy-to-Read Information" (*http://www.disabilityrightsfund.org*) or "Web Content Accessibility Guidelines" (*http://www.w3.org*). In principle, these recommendations can be applied to any language.

Text simplification may involve simplifying lexical and syntactic phenomena, by substituting words that are more frequently used, and by breaking down and changing the syntactic structure of the sentence. As a result, it is expected that the text can be more easily understood (Siddharthan, 2003; Max, 2006). It may also involve dropping parts or full sentences and adding some extra material to explain a difficult point (Petersen and Ostendorf, 2007).

In this Section we focus on the syntactic structure of a text to maximize the comprehension of written texts through the simplification of their linguistic structure by performing a very simple experiment. The produced sentence would be grammatically correct and it would definitely be easier to read and understand.

### 6.1.1 Related Work to Text Simplification

Existing text simplification systems can be classified along three axes: the type of system, rule-based or machine learning-based, the type of knowledge used to identify the need for simplification, and the goals of the system.

There are a few rule-based systems for text simplification (Chandrasekar, Doran, and Srinivas, 1996) or (Siddharthan, 2003), focusing on different

readers (poor literate, aphasic, etc). These systems contain a set of manually created simplification rules that are applied to each sentence. These are usually based on parsed structures and limited to certain simplification operations. Siddharthan (2003) proposed a syntactic simplification architecture that relies on shallow text analysis and favours time performance. The general goal of the architecture is to make texts more accessible to a broader audience. Max (2006) applied text simplification in the writing process by embedding an interactive text simplification system into a word processor. At the user's request, an automatic parser analyzes an individual sentence and the system applies handcrafted rewriting rules. This system requires human intervention at every step.

There are also data-driven systems that, on the other hand, can learn from a corpus the relevant simplification operations and also the necessary degree of the simplification for a given task (Petersen and Ostendorf, 2007). They addressed the task of text simplification in the context of second-language learning. A machine learning approach to simplification was proposed using a corpus of paired articles in which each original sentence does not necessarily have a corresponding simplified sentence, making it possible to learn where writers simplified sentences.

Some language technology systems attempt to simplify documents for various purposes. A variety of simplification techniques have been used, for example substituting uncommon words for common words (Devlin and Tait, 1998), activising passive sentences and resolving references (Canning, 2000), reducing multiple-clause sentences to single-clause sentences (Chandrasekar and Srinivas, 1997; Canning, 2000; Siddharthan, 2002) and making appropriate choices at the discourse level (Williams, Reiter, and Osman, 2003).

### 6.1.2 Dependency Based Text Simplification

We present here a rule-based syntactic simplification system for Spanish. We follow rather a small subset of the whole set of guidelines to define our rules, mainly:

- Use short sentences.
- Do not try to express more than one idea or theme in each sentence.

Our system uses as input a dependency parsed tree for a given sentence and it is therefore limited to a simplification operation applied to the dependency trees. Our system prunes the tree focusing on a certain set of dependency labels, which is the one included in the Spanish treebank. We tested our proposed system over the Spanish treebank used in the CoNLL-X Shared Task (introduced in Section 2.2 and following the data format shown in Section 1.1.2), producing a simplified version of the sentences.

In the following we show why with our methods it is possible to produce a simplified version of the sentences showing an example and several conclusions.

### Dependency Tree Pruning

Our idea was to prune the tree based on the syntactic dependency labels, after analyzing the set of possible tags we were wondering which dependency label is the most appropriate to be removed, we finally focused on a small subset of 3 different dependency labels:

- "CC" (circumstantial object).
- "CD" (direct object).
- "CI" (indirect object).

The reason is mainly that this subset appears in most of the sentences, which was the only way to make a quite aggressive simplification in which we can observe some results. After several checks, we concluded that the only tag that could be deleted without losing the main information of the sentence is the "CC" tag. It expresses complementary information about an action, like *when, where, how*, and *why*. But this "CC" tag never reports about *who* or *what*. Removing the "CC" tag, we are not always loosing the information about *when* or *where* because this kind of information is not always depending on the verbs and the "CC" tags. However, it is also true that sometimes the information removed may be absolutely needed, and therefore the rules used by our system might take this into account.

In the following subsection we present our algorithm that removes the "CC" label, and the subtrees related to this label, from sentences tagged with dependencies and produces a simplified version of the sentence.

### Pruning Algorithm

We implemented an algorithm that takes the dependency tree in the CoNLL Data Format and returns a plain text with the simplified sentence.[1] If the dependency tree is well-formed, meaning that it does not have inconsistencies or cycles, or at least it is correctly tagged for the tags that our algorithm takes into account, the resulting sentence must be grammatically correct.

The algorithm runs through the dependency tree and it makes the following steps:

1. The algorithm removes all the nodes that have as dependency tag the "CC" tag.

[1] This algorithm can only work with the annotation provided in the Spanish treebank.

2. The algorithm removes all the nodes that have as parent the node removed in 1. The algorithm iterates in 2 while there are more nodes that have had a parent removed. In this way the algorithm removes the whole subtree that depends on the root of the already mentioned subtree.

3. It generates a plain text sentence by removing all the morphologic and syntactic information of the dependency tree, which basically means to remove the annotation provided and keep only the FORM column.[2]

Figure 6.1 shows a very easy example of the sentence: *Tocó la vieja pared con cuidado* (in English, *He/She touched the old wall carefully*). The resulting sentence must be: *Tocó la vieja pared* (in English, *He/She touched the old wall*). Our algorithm removes the information about how he/she touched it, which is not needed to understand the main meaning of the sentence.

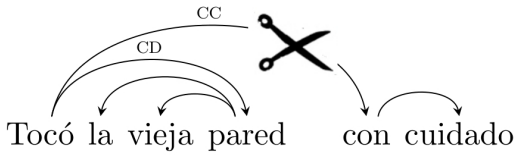



Figure 6.1: Pruning a dependency tree for the sentence: *Tocó la vieja pared con cuidado*, (in English, *He/She touched the old wall carefully*).

Here we show another example, the first sentence (1) is an original version of a complex sentence included in the Spanish treebank. The second one (2) is the output of our pruning tree algorithm for the same sentence:

1. *Tocó el familiar bulto con cuidado, recorriendo sus aristas con las yemas de los dedos, contemplando la imagen que le devolvía el espejo y pensando que todo aquello ya no tenía remedio, que nada podía hacer ya por su cara, ni por su pecho, por esas piernas que no veía, pero sabía tan huesudas y separadas como las patas de un pollo mojado, y por esa carne blanquecina, fofa, que comenzaba a acumularse en torno a su cintura, a descolgarse hacia abajo arrastrando en su vértigo un ombligo progresivamente hondo, para añadir una nueva vejación, la de los años, a un cuerpo condenado de antemano, desde antes de existir, a ser feo. [She touches the familiar shape carefully, following its edges with her fingertips, looking at the image returned by the mirror and thinking that everything was hopeless, there was nothing she could do for her face, her chest, for those legs that she did not even see, but she*

[2]See Section 1.1.2

*knew that they were so bony and separated as wet chicken legs, and for that white and flabby flesh which began to accumulate around her waist, falling down dragging in her vertigo a very deep belly, to add a new vexation of years to a body doomed, even before its existence, to be ugly.]*

2. *Tocó el familiar bulto, recorriendo sus aristas, contemplando la imagen que le devolvía el espejo y pensando que todo aquello no tenía remedio, que nada podía hacer. [She touches the familiar shape carefully, following its edges with her fingertips, looking at the image returned by the mirror and thinking that everything was hopeless, there was nothing she could do.]*

As we can observe in the example, the simplified version is much easier to read and it keeps the main information of the original sentence.

#### Overall Statistics in the Treebank

In this subsubsection we show the statistics of the spanish treebank before and after applying our algorithm, by simplifying the whole corpus. We therefore applied the algorithm sentence by sentence. The Spanish treebank has 3,512 sentences, and the algorithm simplified in 2,737 sentences (77.93%). The algorithm did not simplify the whole corpus, because sentences that do not have a "CC" tag were not simplified. The results of the experiment are given in Table 6.1 which shows the number of wordforms, the average sentence length and the longest sentence length of the original corpus and the simplified corpus.

| | Original | Simplified |
|---|---|---|
| Total Wordforms | 95,028 | 58,415 |
| Average SL | 27.06 wf | 16.63 wf |
| Longest SL | 143 wf | 94 wf |

Table 6.1: Overall Statistics in the corpus before and after the simplification considering sentence length (SL).

### 6.1.3 Evaluation

In this Section we present how we evaluated the text simplification system in order to determine the usefulness of our method.

### Evaluation Design

The evaluation is based on two surveys about the outcomes of the system. The first evaluation measure consists in a group of computer science students and researchers. The second evaluation measure consists in a group of children between ten to eleven years old, which in this case it is a group objective evaluation, because the children belong to one of the groups in which this kind of systems may be needed. They have more problems than the adults, when they have to read complex sentences.

We surveyed the first group of 20 people, about how good was the text simplification made by our algorithm. They all had university studies and they all spoke Spanish as their native language. None of them knew how the simplification algorithm works. We selected 20 sentences from the Spanish treebank. We showed them the whole sentence and the simplified sentence, then we asked them the following questions, they had to answer just "yes" or "no":

- Q1: *Is the main idea of the sentence retained?*
- Q2: *Was all the removed information unnecessary?*
- Q3: *Have only minor details been deleted?*
- Q4: *Do you understand the simplified sentence better than the normal sentence?*

Q2 and Q3 are quite similar, however, it is worth mentioning that in this kind of surveys is good to have related or similar questions, with close semantics, but with slight differences in meaning. However, there are some differences between Q2 and Q3. The negative answer to Q3 indicates lower quality of the compressed sentence, but Q2 is more general about the idea that we loose some information.

As a second evaluation measure, we decided to carry out an evaluation with a group objective, consisting in 24 children between ten to eleven years old. We selected 20 sentences from the corpus, we showed them the simplified version and the original version, they had to answer 'yes' or 'no' to the following question for each sentence: *Do you understand better the simplified sentence than the normal sentence?*

### Results and Discussion

Table 6.2 shows the results of the evaluation made by the group of adults. In the table we show the answers 'yes' or 'no' for each question.

The first question, Q1: *Is the main idea of the sentence kept?*, is the most important one. The survey gave us 67.58% of people that say "yes"

| Question | YES | NO |
|---|---|---|
| Q1 | 67.58% | 32.42% |
| Q2 | 27.66% | 72.34% |
| Q3 | 46.72% | 53.28% |
| Q4 | 60.76% | 39.24% |

Table 6.2: Results obtained in the survey for adults.

for the selected sentences in question Q1. We can conclude that most of the people thought that in most of the sentences the main idea, and the meaning of the sentence were preserved. Most of the sentences in which the people answered "no" were very long and our system made very aggressive simplifications that lead to some part of the meaning were missing.

If we focus on question Q2: *Was all the removed information unnecessary?*, people thought that not all information was dispensable. It is probably because our algorithm made very aggressive simplifications in many cases. Looking at questions Q1 and Q2, we can see that most people feel that we are loosing some information but they think that the overall meaning was preserved.

Concerning the third question Q3: *Have only minor details been deleted?*, if we look at the results we can conclude that in some of the sentences where we loose some data, we are not loosing the most important information.

If we focus on the last question Q4: *Do you understand better the simplified sentence than the normal sentence?*. This question asks about how well the people understand the simplified version compared to the normal sentence. Most of the people thought that the simplified sentences are easier to read, however a significant minority (a bit less than 40%) did not. It is important to notice that some of the sentences are not really difficult to read in the original version and because of that, some people answer "no" to this question.

Finally, as a conclusion of the experiment, we see that most of the people felt that the main idea of the sentences was preserved, which is one of our goals, and they also thought that the simplified version is easier to read and understand than the original version which is our second goal.

The results of the survey on children are presented on Table 6.3. We had 240 answers, 20 answers for each sentence. The children answered "yes" in 125 of the 240 cases. Therefore, we have 52.08% of children who believed that the simplified sentence was easier to read than the original version.

We can see the differences between the 4th question Q4 in the first evaluation measure, and the results given by the survey in this evaluation measure. In question Q4 people are not in the group objective, so they can not say that they understand better the sentences because they (should) understand them at the same level. In this second evaluation measure the

| Children | YES | NO |
|---|---|---|
| 24 | 52.08% | 47.92% |

Table 6.3: Results obtained in the survey for children.

children may have some problems to understand the sentences properly, so our system can help them to understand the information better. In fact children may have difficulty in understanding even the simplified sentence because they are not able to read some difficult concepts that are presented in the original version and the simplified version.

We can conclude that taking into account both evaluations, our system helps to understand the sentences better, which was our main goal.

### 6.1.4 Conclusions

The potentialities of text simplification systems for education and people with difficulties in understanding are evident. For students, it is a first step towards more effective learning. For people with poor literacy, we see text simplification as a first step towards social inclusion, facilitating and developing reading and writing skills making them able to interact in society. The social impact of a good text simplification system is something worth noting.

The presented system is a first approximation to a complete automatic system that runs through dependency trees and returns a simplified version of the sentence parsed. We can conclude that it is possible to simplify correctly texts using dependency parsing, in the particular case of Spanish. The simplified sentence is grammatically correct due to the potential usefulness that dependency structures provide. But on the other hand, choosing any label and using dependency parsing, the algorithms may make aggressive simplifications. However, it is worth emphasizing that we made a simple version of the algorithm and we only focused on the dependency trees, which we believe it is good to show the potentialities of dependency parsing in this task.

It is worth noting that we would be able to apply our method if we use dependency structures annotated in the same way as they are annotated in the Spanish treebank. This is something easy to do by applying a parser generator, such as MaltParser or MSTParser, trained over the whole Spanish treebank and annotating new sentences that are about to be simplified. However, we should take into account that these parsers generators (as we saw during Chapters 3, 4 and 5) are not perfect, and they produce models with a high accuracy but they are not error free, this is why the simplifications made over these new hypothetical sentences should be observed under this perspective.

## 6.2 Inferring the Scope of Negation for English via Dependency Analysis

Generally speaking, negation turns an affirmative meaning into a negative one (*I want / I do not want*) and it may indicate whether it is a fact or not. It is also a complex phenomenon in natural languages and it has been an active research topic for decades. Researchers have approached this topic from both linguistic and philosophical perspectives (Horn, 1989). In most cases, negation involves a negation cue and a negated syntagma containing one or more words that are within the scope of this negation. In the following example: *"There is no detectable effect on leg segmentation"*, *'no'* is the negation cue used to denote that the following concept is negated, being *"detectable effect on leg segmentation"* the negated syntagma. There is also another concept which is the negated event, that indicates the intended negated concepts from a given negative action, and it is also worth detecting it.

Nowadays, negation detection is an emergent task in natural language processing. Detecting uncertain and negative assertions is essential in most text mining tasks where, in general, the aim is to derive factual knowledge from textual data. For instance, in text mining extracted information that is within the scope of negation should either be discarded or presented separately from factual information. It can be applied to several related areas, such as opinion mining (Carrillo de Albornoz et al., 2011), (Councill, McDonald, and Velikovich, 2010).

Moreover, negation is commonly seen in clinical documents and it is an important source of low precision in automated indexing systems. As it is evidenced in Chapman's (2002) work. When querying large medical free–text databases, the presence of negations can yield numerous false–positive matches, because the medical personnel is trained to include pertinent negatives in their reports. In a search for *fracture* in a certain radiology reports database, 95 to 99 percent of the returned reports would state *"no signs of fracture"* or words to that effect (Chapman et al., 2002). Therefore, to increase the utility of indexing medical documents, it is necessary to acknowledge whether words have been negated or not.

In this Section we present a system that annotates the scope of negation in sentences written in English, making use of a simple technique:[3] first a manually-defined set of keywords is matched, then an algorithm marks the range of the scope within the sentence using unlabeled dependency syntactic structures. We evaluated the system with an established corpus annotated with the scope of negations, Bioscope (Vincze et al., 2008), which is a corpus annotated with negations. Finally, in Section 6.2.6 we show how we adapted

[3]The system presented here can be accessed and downloaded via http://minerva.fdi.ucm.es:8888/ScopeTagger

the system to work with a different corpus in order to participate in a Shared Task.

### 6.2.1 Related Work

Nowadays there are two main kinds of systems that work with negation: systems that detect wordforms affected by negations, and (more recent) systems that classify the whole scope of negations, which is a more difficult task. Our system is classified in the second kind of systems. There are also some approaches that try to extract negated events, such as (Sarafraz and Nenadic, 2010), we also extracted negated events in our participation in the *SEM Shared Task, see Section 6.2.6.

For the biomedical domain there is plenty of research studying negation and finding how to detect it. For instance, Chapman et al. (2001) detected negations and identified medical terms affected, by means of the well known and simple regular expression algorithm called NegEx. It achieves 84.5% precision and 77.8% overall recall over 400 randomly selected sentences. In a similar way Mutalik et al. (2001) recognized negated patterns in biomedical texts by using a training set of 40 medical documents; the set was manually inspected and used to develop a rule–based system (Negfinder), able to recognize a set of negated patterns in texts. They showed very well evaluation results, verified by human interaction, yielding 95.7% recall and 91.8% precision. Also, Huang and Lowe (2007) implemented a hybrid approach to an automated negation detection system. They combined regular expression matching with grammatical parsing, to check the limits in automatically detecting negations in clinical reports. Their approach identified negated phrases with 98.6% precision and 92.6% recall over a test set of 120 reports.

On the other hand, there are systems that infer the scope of negation. This is a more difficult problem, because it involves determining the words that are within the scope of a negation cue, where to open the scope, and finally, where to close it. One of the main works has been carried out by Morante's team (Morante, Liekens, and Daelemans, 2008; Morante and Daelemans, 2009) in which a machine learning approach for the biomedical domain is shown. The system was evaluated with the Bioscope corpus and their results were: 80.11% overall precision and 78.44% recall in finding scopes of negation.

In 2010, a Workshop on Negation and Speculation in Natural Language Processing (Morante and Sporleder, 2010) was held in Uppsala, Sweden. One of the main objectives was to find the lexical aspects of negation to define how it could be modeled, in order to analyze how it can be used in information retrieval applications and natural language processing. Most of the approaches presented in the Workshop were in the biomedical domain, which is probably the most studied one in negation detection. An interesting

paper for our work was presented in that workshop by Councill et al. (2010). They used the Bioscope corpus to evaluate their scope finding system, that is based on conditional random fields based on dependency parsing features, and their results were 78.2% recall and 81.9% precision.

Later, Zhu et al. (2010) presented a unified framework for scope learning by means of shallow semantic parsing, evaluating it with the Bioscope corpus. They divided the process into three main steps and they carried out the evaluation considering *golden cues* (which means that their system does not need to find where the cue is and which one it is), and *golden trees* (which means that their system does not need to find what the correct tree is, because it is given). They also reported their results when the system should predict correctly the tree and the cue. Their results, without using golden cues were 72.53% recall and 72.24% precision. When the *golden cue* was given, they were notably higher reaching almost 90%. This means that such kind of system in synergy with an accurate cue classifier could be an interesting option to achieve very high results.

In Table 6.4, the results of the Morante et al. system, the Councill et al. system and the Zhu et al. system are shown, in order to infer the scope of negation using the Bioscope corpus as a standard of measuring. These systems are the ones that performed the evaluation in a most similar way to ours.

| **System** | **Evaluation Set** | **Recall** | **Precision** |
|---|---|---|---|
| (Morante and Daelemans, 2009) | Whole **Bioscope** | 78.44% | 80.11% |
| (Councill, McDonald, and Velikovich, 2010) | **Bioscope** Full Papers | 70.8% | 80.8% |
| (Councill, McDonald, and Velikovich, 2010) | Reviews | 78.2% | 81.9% |
| (Zhu et al., 2010) | Whole **Bioscope** | 72.53% | 72.24% |

Table 6.4: General results of systems that infer the scope of negation.

And finally, another interesting approach is presented by Agarwal and Yu (2010). Their system uses conditional random fields in order to classify the scope of negation cues. They achieved an F1-score of 98% and 95% on detecting negation cue phrases and their scope in clinical notes, and an F1-score of 97% and 85% on detecting negation cue phrases and their scope over the Bioscope corpus, their system can be downloaded from the web.[4]

### 6.2.2 Bioscope Corpus

Bioscope[5] (Szarvas et al., 2008) is an open access corpus, annotated manually with the scope of negation for the biomedical domain.

The Bioscope corpus contains more than 20,000 sentences, divided in three different collections:

[4] http://snake.ims.uwm.edu/negscope/index.php

[5] www.inf.u-szeged.hu/rgai/bioscope

- Clinical Documents. It consists of 1,954 documents (6,383 sentences) containing clinical reports with the impressions of radiologists.
- Scientific Papers. It consists of 9 different scientific papers (2,670 different sentences).
- Scientific Papers Abstracts. It consists of 1,273 abstracts (11,871 sentences) extracted from the Genia corpus (Kim et al., 2003).

Table 6.5 shows the number of documents, sentences, negation sentences and negation cues for each collection, as well as the average sentence length and the percentage of scopes to the right and to the left in the Bioscope corpus taking only into account sentences containing negation signals.

| | Clinical | Papers | Abstracts |
|---|---|---|---|
| Documents | 1,954 | 9 | 1,273 |
| Sentences | 6,383 | 2,670 | 11,871 |
| Negation Sentences | 863 | 339 | 1597 |
| % Negation Sentences | 13.55 | 12.70 | 13.45 |
| Negation Cues | 877 | 389 | 1848 |
| Av. Sentence Length | 7.73 | 26.24 | 26.43 |
| %Scopes to the right | 97.64 | 81.77 | 85.70 |
| %Scopes to the left | 2.35 | 18.22 | 14.29 |

Table 6.5: The statistics of the Bioscope corpus considering only sentences with negations.

### 6.2.3 Negation Scope Finding

Our system consists of two algorithms: the first one is responsible for inferring words affected by the negative operators (cues) traversing dependency trees and the second one annotates sentences within the scope of negations. This second algorithm consists of a set of rules that have been built making use of a development set, which was extracted from the Papers collection of the Bioscope corpus, more concretely, this development set was formed by the first 10% of the sentences that appear in the Bioscope Scientific Papers Collection (this set of sentences is removed in the Evaluation presented in Section 6.2.4).

In the first algorithm our system traverses a dependency tree, searching for negation cues to determine the correct scope over the tree. Our contribution lies in the identification of the scope, which is not explicit in the dependency tree. We selected Minipar parser (Lin, 1998) to develop the present experiment, which is a rule–based dependency parser capable of perform high accuracy unlabeled parsing. We selected Minipar because

we only need unlabeled parsing and there is no need to train the system collecting annotated corpora. However, it is also worth noting that by using a data-driven dependency parser we could get a different point of view of the task.

Therefore, our system works as follows: a parse given by Minipar and a negation cue lexicon (as described below), are the inputs for an Affected Wordforms Detection Algorithm. Then, a Scope Finding Algorithm acts on the set of negated nodes by using a passive voice detector returning an annotated sentence with the scope of negation. Figure 6.2 shows the system architecture and in the following subsections we describe each component in turn.

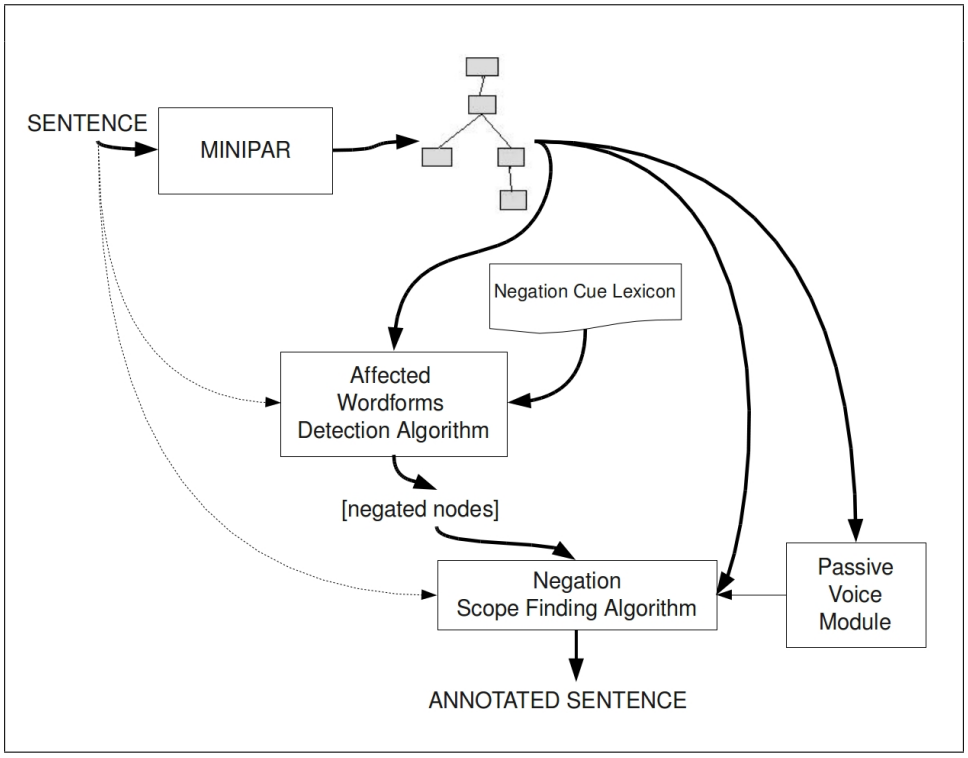



Figure 6.2: System architecture showing each module that conform the whole system.

### Negation Cue Lexicon

To determine the scope of negation, first of all a set of negation cues must be established. We considered the guidelines presented in (Morante, 2010) to set up our negation cue lexicon and also the work done by Mutalik et al. (2001) in which some of the negation cues were described.

The lexicon used in our system is shown in Table 6.6. This lexicon only

contains the lemmas of each wordform, but our system is able to parse not only the lemma but all kind of verb forms.

| | | | |
|---|---|---|---|
| not | no | neither..nor | none |
| discard | rule out | fail | avoid |
| absence | lack (v) | lack (n) | without |
| unable | rather than | absent | cannot |

Table 6.6: Lemmas of the Bioscope negation cues contained in our Negation Cue lexicon.

These negation cues are the ones selected to develop the present work, but similar analyses can be accomplished with a different set.[6]

### Affected Wordforms Detection Algorithm

We implemented an algorithm that takes the dependency tree for a sentence returned by the dependency parser and returns for each negation cue a set of words affected by the cue. It uses the lexicon of negation cues presented in the previous section.

The algorithm traverses the dependency tree of a sentence, and it carries out the following steps:

1. It detects all the nodes that are contained in the lexicon of negation cues.

   - If the negation cue is a verb, it is marked as a negation cue.
   - If the negation cue is not a verb, the algorithm marks the verb (if exists) affected by it as a negation cue. In this way, the words that depends on the verb are affected by the negation cue.

2. For the rest of nodes, if a node depends directly on any of the ones previously marked as a negation cue, the system marked it as negated. Moreover, the negation is propagated from the cue word through the dependency graph until finding terminals, so wordforms that are not directly related with the cue are detected too.

The algorithm finally generates a set of nodes containing the wordforms within the scope of negation cues involved in the sentence.

[6]See Section 6.2.6 in which we show modifications of the present system by changing the lexicon of cues making it able to annotate sentences from a different domain.

### Scope Finding Algorithm

This second algorithm is implemented using a subset of the Scientific Papers collection of the Bioscope corpus (first 10% of the sentences containing negations) as development set in order to give shape to the rules involved. We selected the Scientific papers collection to come up with the rules because it contains a greater variety of sentences and several negation cues, which makes it much more diverse.

The algorithm works with the set of words returned by the Affected Wordforms Detection Algorithm, shown above, and the dependency tree given by the dependency parser in order to annotate sentences with the scope of negation, inferring where the scope must be opened and where it must be closed.

Note that when the scope is opened to the right of the negation cue, the scope of negation leaves the subject out. This correspond to sentences in active voice and they are the most frequent ones. Additionally, there are some cases in which the scope is opened to the left of the negation cues. The most frequent one is the passive voice. As shown in (Szarvas et al., 2008), passive voice is an exception in the way of tagging sentences in Bioscope. In this case the subject is marked within the scope of negation, because if the sentence had been written in active voice, it would be the object of a transitive verb.

The first thing to do is to decide where to open the scope of a new negation, which, in the case of Bioscope, is inherently related to the voice of the sentence: if the sentence is in passive voice the scope must be opened to the left of the negation cue and if the sentence is in active voice, the scope must be generally opened to the right of the cue.

Therefore, we considered that the Scope Finding Algorithm must be divided in two main processes: first, to detect if the sentence follows a negated passive voice structure or not, and second, to annotate the sentence with the scope of negation, or scopes if there is more than one (which is an usual situation).

Our system decides that a sentence is in passive voice if the following conditions occur:

- It contains a transitive verb, such as, *show, consider, see, use, detect*, etc.
- It follows one of the patterns shown below:
    1. *modal verb + <u>not</u> + be + past participle.*
    2. *am/is/are/was/were + <u>not</u> + past participle.*
    3. *have/has been + <u>not</u> + past participle.*

Once our system has decided if the sentence is in passive voice or not, the Scope Finding Algorithm iterates through the sentence, token by token

and applies a set of rules about the scope opening and closing. The rules are applied in the order presented below and it only applies one rule for each token.

1. Scope opening:

   **a.** If the token is contained in the set of nodes marked as negated by the Affected Wordforms Detection Algorithm and the scope for the cue involved is not open: the system opens the scope at the token and establishes that the scope for the cue involved is already opened.

   **b.** If the token is a negation cue and the sentence is in passive voice: the system goes backward and opens the scope just after the last punctuation symbol or the last clause boundary (such as but). The system opens and closes the cue at the token.

   **c.** If the token is a negation cue and the sentence is not in passive voice: the system opens the scope just before the token. The system opens and closes the cue at the token.

2. Scope closing:

   **a.** If the token is a punctuation symbol, followed by some wordforms that indicates another clause, such as *but*: the system closes the scope just after the token.

   **b.** If the token is any wordform and all the nodes that are marked as negated for the negation cue are already included in the scope: the system closes the scope just before the token.

   **c.** If the token is the end of sentence: the system closes the scope at the end of the sentence.

3. Adding words to the sentence: if none of the previous rules has been applied the token is added to the annotated sentence.

At this point, the system has computed the scope (scopes) of the negation (negations) for a given sentence, by inferring which nodes pertain to that scope (scopes) from the node (nodes) marked as negated. Figure 6.3 illustrates the processing of the following sentence: *The reason why the two other families were not detected is more complex*. In the figure, the potential usefulness of the dependency syntactic structure to infer the scope of negation is evidenced. As shown in the Figure, the rule applied to open the scope and open and closes the cue is *1b*. Finally, the rule applied to close the scope is *2b*, wherein the system closes the scope because there are no more wordforms marked as affected by the algorithm described in Section 6.2.3.

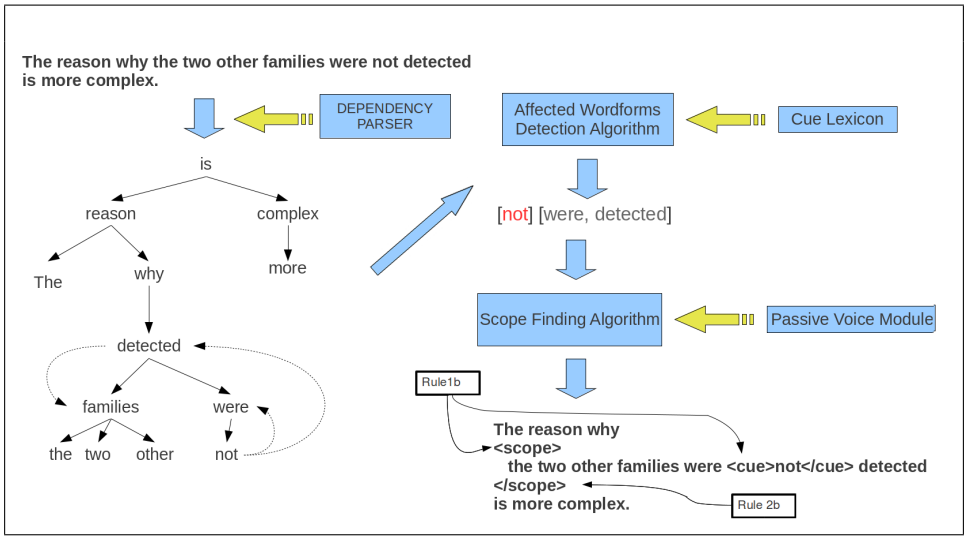



Figure 6.3: The processing of a sentence by our system.

### 6.2.4 Evaluation

Here, we present the evaluation performed to test our system: the design of the evaluation as well as the evaluation metrics, the results considering these metrics, and finally, we compared our results with other published approaches.

#### Evaluation Design

We selected Bioscope as the corpus for our evaluation because when the system was developed there were no other corpora annotated with the scope of negations. In Section 6.2.6 we show our participation in a competitive task in which we had to modify the present system and we were able to evaluate the system in a different domain with a different corpus.

It is worth emphasizing that the evaluation is carried out over the output of the Scope Finding Algorithm, which is the output of the whole system, it uses the Affected Wordforms Detection Algorithm.

Our first step in order to get the results was to select the sentences containing negations in the three collections of Bioscope, considering that for the Scientific Papers collection we must remove 10% of these sentences because this is the data set used to develop the rules of the algorithm presented in Section 6.2.3. Therefore, we evaluated our system with 100% of the clinical sentences containing negations, 90.0% of the paper sentences containing negations and 100% of the abstract sentences containing negations.

The following evaluation measures were used:

$$P(Precision) = \frac{Tokens \quad correctly \quad negated \quad by \quad our \quad system}{Tokens \quad negated \quad by \quad the \quad system}$$

$$R(Recall) = \frac{Tokens \quad correctly \quad negated \quad by \quad our \quad system}{Tokens \quad negated \quad in \quad the \quad collection}$$

To balance the results in recall and precision, we used micro F1.

$$F1 = \frac{2PR}{P+R}$$

Additionally, we evaluate our system with the percentage of correct scopes (PCS), and the percentage of correct negation cues (PCNC).

$$PCS = \frac{Correct \quad Scopes \quad annotated \quad by \quad our \quad system}{Scopes \quad annotated \quad in \quad the \quad collection}$$

$$PCNC = \frac{Correct \quad negation \quad Cues \quad annotated \quad by \quad our \quad system}{Negation \quad Cues \quad annotated \quad in \quad the \quad collection}$$

By using all these measures we are considering not only a token–based evaluation but a whole scope classification measure, that really shows how good the system is in classifying the scope of negation in sentences.

### Results and Discussion

When parsing the three collections of Bioscope, our system obtained the results given in Table 6.7.

Note that we did scope identification with automatic cue recognition, so the input of our program, as shown in Section 6.2.3, is the sentence without any extra information.

| Collection | Precision | Recall | F1 | PCS | PCNC |
|---|---|---|---|---|---|
| Papers | 73.49% | 80.70% | 76.93% | 56.43% | 91.15% |
| Abstracts | 84.92% | 84.03% | 84.48% | 68.92% | 95.56% |
| Clinical | 95.83% | 90.58% | 93.13% | 89.06% | 94.82% |

Table 6.7: Results of our work, when evaluating it with the three collections of Bioscope.

The different results could be explained as follows. The main reason for the better results that our system achieves when annotating the sentences from the Clinical Reports collection is that the sentences contained in the clinical reports collection followed very easy syntactic structures and most of them contain the scope to the right. The average sentence length in

the clinical reports collection is 7.73 wordforms, while in the papers and abstracts collections is more than 26 wordforms (Morante and Daelemans, 2009). Moreover, a lot of sentences in the abstracts and papers collections contain more than one negation cue, which are more difficult to parse than those having only one. Finding the scope of negation in the simpler sentences of the clinical reports collection is easier than finding it in the sentences of the abstracts and papers collections.

In a similar way, the results for the abstracts collection are better than the ones for the scientific papers collection. One possible reason for this is the simplicity that usually characterizes abstract sentences. Sentences in an abstract are usually easy to understand, the writer commonly shows the main ideas of what is explained below with more simple syntactic structures.

Analyzing the sentences with mistakes, we found that there are two main reasons for these errors:

- Minipar, as any other dependency parser, is not error free, being able to cover about 79% of the dependency relations. If the dependency tree returned by Minipar is not correct, our system is not able to infer accurately the scope of negation in the sentence. In some cases we found that the tree is incomplete, and some information is missed. In these rare cases our system does not have enough information to decide how far the scope must be annotated. In most of these cases the cue is not correctly found and if it is, the scope probably closes incorrectly.

- There are some negation cues that are not always considered as negation cues, such as *negative*. This fact is evidenced in Morante's work (Morante, 2010). Due to the characteristics of our system, we must define the negation cue lexicon at the beginning. A semantic module is the obvious suggestion to tackle these special cases, because we could be able to infer the negative semantics of a single word that is not always acting as a negation signal avoiding the noise produced by an initial static decision.

### Comparison with other Systems

In this Section, we show an approximate comparison with some of the systems of the state of the art. We compare our results with the machine learning approach of Morante and Daelemans (2009), the shallow semantic parsing approach of Zhu et al. (2010) and the dependency system of Councill et al. (2010). The main comparison is shown in Table 6.8 where we show the precision, recall, F1, PCS and PCNC with other state of the art approaches. As evidenced in the results, our system performs very well for all the results with the exception of PCNC, in which the effect of our static and small lexicon of cues causes noise in the performance.

It is important to notice that in this table we show the results of Zhu's and Morante's systems when using automatic cue recognition, as we did in our system. Therefore, we are not reporting their results when using neither *golden cues* nor *golden trees*, which are much higher. In addition, for Councill's system only results for the scientific papers collection are shown because it is the only collection in which they published results.

| Collection | System | Precision | Recall | F1 | PCS | PCNC |
|---|---|---|---|---|---|---|
| Papers | Our Results | 73.49% | **80.70%** | **76.93%** | **56.43%** | 91.15% |
| | Morante et Al. | 72.21% | 69.72% | 70.94% | 41.00% | **92.15%** |
| | Zhu et Al. | 56.27% | 58.20% | 57.22% | – | – |
| | Councill et Al. | **80.80%** | 70.80% | 75.50% | 53.70% | – |
| Abstracts | Our Results | **84.92%** | **84.03%** | **84.48%** | **68.92%** | **95.56%** |
| | Morante et Al. | 81.76% | 83.45% | 82.60% | 66.07% | 95.09% |
| | Zhu et Al. | 78.24% | 78.77% | 78.50% | – | – |
| Clinical | Our Results | **95.83%** | **90.58%** | **93.13%** | **89.06%** | 94.82% |
| | Morante et Al. | 86.38% | 82.14% | 84.20% | 70.75% | **97.72%** |
| | Zhu et Al. | 82.22% | 80.62% | 81.41% | – | – |

Table 6.8: Results of our work, evaluated with the three collections of Bioscope and compared with the systems of Morante et Al., Zhu et al. and Councill et Al.

Morante's system is based on machine–learning. In contrast, our system was constructed using as development set a subset of the sentences presented in the papers collection, as it is described in Section 6.2.3. Thus, while we tested our system with the Bioscope corpus (with the exception of the first 10% of the development set of Papers), Morante et al. performed 10–fold cross validation experiments over the abstracts collection. And, for the other 2 collections, they trained with the abstracts set and tested with the corresponding collection. This fact affects the results, but we tried to make the results as comparable as possible.

This is why the Morante et al. results are much more comparable to ours in the case of the Abstracts collection. In the same way, Councill et al. results carrying out the experiment only with papers is much more directly comparable to our results because they get paper sentences to carry out the training. As shown in Table 6.8, we can observe how these 2 cases produced similar results to ours. The Councill et al. system seems to be very competitive because the original task of this work was to annotate sentences to solve a sentiment analysis task, nevertheless, their results with Bioscope were very good.

In Table 6.9 we show the percentage of correct scopes (PCS) per negation cue, for negation cues that occur 10 or more times in each collection present in Bioscope. We compare our results with the ones published by Morante and Daelemans (2009), which is the same system studied in Table

6.8. Negation cues with a lower PCS have a higher percentage of scopes to the left (*absent, unable*). In this case we consider all the test set including the data seen (in the algorithm construction) in the Papers collection, that conforms the 100% of the paper sentences containing negations in order to obtain the same numbers as Morante et al. system, therefore, it is worth to mention that this column of data should be observed under this perspective.

| | Abstracts | | | Papers | | | Clinical | | |
|---|---|---|---|---|---|---|---|---|---|
| | # | **Mor.** | **Our** | # | **Mor.** | **Our** | # | **Mor.** | **Our** |
| **absence** | 57 | 56.14 | **71.93** | – | – | – | – | – | – |
| **absent** | 13 | 15.38 | **38.46** | – | – | – | – | – | – |
| **cannot** | 28 | **42.85** | 28.57 | 16 | 50.00 | 50.00 | – | – | – |
| **fail** | 57 | 63.15 | **85.97** | 13 | 38.46 | **53.84** | – | – | – |
| **lack** | 85 | **57.64** | 52.94 | 20 | 45.00 | **50.00** | – | – | – |
| **neither** | 33 | 51.51 | **72.72** | – | – | – | – | – | – |
| **no** | 207 | 73.42 | **81.64** | 44 | 50.00 | **54.54** | 673 | 73.10 | **89.60** |
| **none** | – | – | – | 10 | 0.00 | **71.42** | – | – | – |
| **not** | 1036 | **69.40** | 66.41 | 200 | 39.50 | **64.50** | 57 | 50.87 | **66.66** |
| **rather than** | 20 | 65.00 | 65.00 | 12 | **41.66** | 25.00 | – | – | – |
| **unable** | 30 | 40.00 | **73.33** | – | – | – | – | – | – |
| **without** | 82 | **89.02** | 79.27 | 24 | 58.33 | **70.83** | – | – | – |

Table 6.9: PCS per negation cue for negation cues that occur 10 or more times in one of the subcorpus and appear in our lexicon of negation cues. The column # shows the number of appearances for each case; the column **Our** shows our system values and Morante's system values are given in column **Mor.**

In this depicted table of results we can see how the Morante's machine learning approach is not able to cover negation signals with a very low frequency in the training set but a higher frequency in the test, as we can see in the results for the negation cue *none* in the papers collection (recall that they used the abstract collection for training and carried out the testing with the papers collection, in this case). Nonetheless, our system classifies the scope of this signal with higher accuracy.

Considering the most frequent negation cues, *not* and *no* (these cues are the ones with the strongest effect on the accuracy at the end), our system beats the results of Morante et Al. in clinical reports and papers collection. However, they beat our results for the cue *not* in the abstracts collection.

Finally, we did not include the Agarwal and Yu's work (Agarwal and Yu, 2010) in the comparison, which achieves an F1-score of **98**% and **95**% on detecting negation cue phrases and their scope in clinical notes, and an F1-score of **97**% and **85**% on detecting negation cue phrases and their scope in biological literature. This approach using conditional random fields presented very high results, but as discussed in the Tutorial Given at the

IJCNLP 2011 conference at Chiang Mai, Thailand,[7] the corpus partitions and the evaluation measures are different, because the sentences without negations in the corpus were also used for the evaluation. Thus, the systems are, at least, not directly comparable, which shows that there are other ways to evaluate this task. Nevertheless, we consider important to mention this work, that includes a website in which is possible to use the system and shows a very interesting approach, and present very robust results as we described in Section 6.2.1.

### 6.2.5 Conclusions

We have presented a system able to infer the scope of negations. From the results of our experiments we can conclude that dependency parsing is a valuable auxiliary technique for negation detection, at least in the particular case of English and the studied case.

We consider that the scope of negation must not always be annotated as continuous. In Bioscope, the scope of negation leaves the subject out, with the exception of passive voice sentences. Nonetheless we consider that the subject must always be considered as a part of the scope. Moreover, when there is an affirmative sentence that affects the subject of a negative passive voice sentence, it is difficult to infer automatically which subject is considered if we follow the Bioscope guidelines. For instance, in the following sentence *"Therefore, TNF-alpha mRNA induction by PMA, like its induction by virus and LPS, [is* ***not*** *primarily mediated by NF-kappa B], but rather is mediated through other sequences and protein factors."*, the scope of negation is in passive voice but the subject is implicit by the word *is*, and it is not directly included in the scope of negation if we follow the annotation guidelines of Bioscope, as it is done in the present work. The same idea is shown in the Figure 6.4.

Thus, we suggest that the scope must be discontinuous in the way of considering other wordforms that in Bioscope are out of the scope, but are directly affected by the negation cue. It can be achieved using a tabular format for the corpus, instead of plain sentences annotated with XML language. As we show in the present work, we decided to evaluate our system with Bioscope, thus our system annotates the sentences in the same way as it is done in that corpus. In Section 6.2.6 we show the modifications done to this system in order to participate in the *SEM Shared Task on resolving the scope and focus of negation, in which the scopes are annotated in a tabular format being able to annotate more complex structures and discontinuous scopes, which makes the task much more complicated.

Observing the results shown in Table 6.9, where we show the PCS per negation cue, an interesting idea though could be a system that uses in

[7] http://www.ijcnlp2011.org/ijcnlp2011/downloads/tutorial/tu3_present.pdf

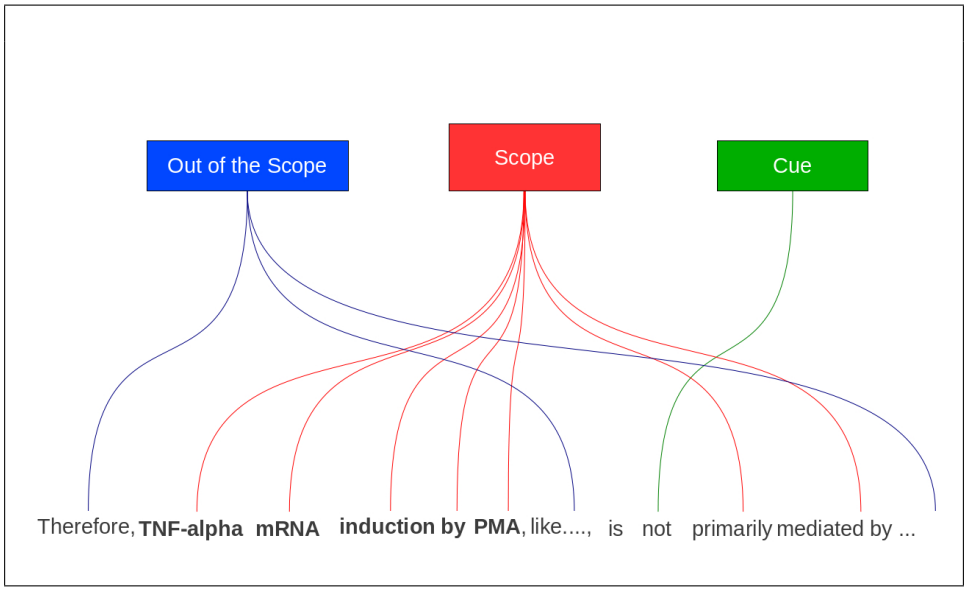



Figure 6.4: Emphasizing the idea of annotating the Scope in tabular format in order to have a more informative annotation.

synergy the results of two systems. By the implementation a voting system that select the best output for the two systems that perform the same computational task, depending, for instance, on the cue involved.

### 6.2.6 The Participation in the *SEM Shared Task Challenge

The *SEM Shared Task[8] (Morante and Blanco, 2012) was based on inferring and classifying the scope and event associated to negations, provided a training and a development corpus based on Conan Doyle's stories (Morante and Daelemans, 2012). In the Shared Task, there were two different tracks:

1. The open track in which researchers could either use their own resources or use the annotation provided in the corpus.

2. The close track in which researchers must use the annotation provided in the corpus.

We presented the system shown above to the Open Track of the Shared Task, modifying the output in order to handle the new structures provided in the corpus, as we show in the present Section.

#### The Conan Doyle Corpus

The corpus used in the Shared Task consists in several chapters of Conan Doyle's stories (Morante and Daelemans, 2012), in which there were different

[8] http://www.clips.ua.ac.be/sem2012-st-neg/

levels of annotation. It is annotated in a similar way as the CoNLL data format[9] corpus by using a tabular format for the scopes of negations. The Figure 6.5 shows an example. The different columns contain the following information:

1. Chapter number and identifier.
2. Sentence counter.
3. Token counter.
4. Form: Word form.
5. Lemma version of the word.
6. Part of speech, using the Penn treebank annotation.
7. Phrase structure parsing, using the Penn treebank annotation.
8. Cue. The form if the token is a cue word, otherwise a null value.
9. Scope. The form if the token belongs to a scope of a negation, otherwise a null value.
10. Negated event. The form if the token belongs to a negated event of a negation, otherwise a null value.

If the sentence has more than one scope, the first scope would be annotated in the columns 8, 9 and 10, the second one would be in the columns 11, 12 and 13 and so on. The subsequent columns contain the same kind of annotation as the columns 8, 9 and 10 (shown in the list above) but with the information related to the new scopes. This kind of annotation permits discontinuous scopes and nested scopes without much problem, which is basically what we emphasized in Section 6.2.5, and it is something that was not possible with Bioscope.

The participants of the Shared Task had at the beginning two different data sets: a development set and a training set, conforming a total amount of 3,899 sentences with 1,056 sentences containing negations. Once the participants submitted their system they evaluated the system with a different data set that was a *blind* test set, which means that the organizers removed the annotation concerning the scope of negation and the negated event, which is basically the output of the participant's systems.

[9] See Section 1.1.2 and Figure 1.1.

```
baskervilles01  8       0       Holmes  Holmes  NNP     (S(S(NP*)       _       _       _       _
baskervilles01  8       1       was     be      VBD     (VP*    _       _       _       _
baskervilles01  8       2       sitting sit     VBG     (VP*    _       _       _       _
baskervilles01  8       3       with    with    IN      (PP*    _       _       _       _
baskervilles01  8       4       his     his     PRP$    (NP*    _       _       _       _
baskervilles01  8       5       back    back    NN      *))     _       _       _       _
baskervilles01  8       6       to      to      TO      (PP*    _       _       _       _
baskervilles01  8       7       me      me      PRP     (NP*)))))       _       _       _       _
baskervilles01  8       8       ,       ,       ,       *       _       _       _       _
baskervilles01  8       9       and     and     CC      *       _       _       _       _
baskervilles01  8       10      I       I       PRP     (S(NP*) _       I       _       _
baskervilles01  8       11      had     have    VBD     (VP*    _       had     _       _
baskervilles01  8       12      given   give    VBN     (VP*    _       given   given   _
baskervilles01  8       13      him     him     PRP     (NP*)   _       him     _       _
baskervilles01  8       14      no      no      DT      (NP(NP* no      _       _       _
baskervilles01  8       15      sign    sign    NN      *)      _       sign    _       _
baskervilles01  8       16      of      of      IN      (PP*    _       of      _       _
baskervilles01  8       17      my      my      PRP$    (NP*    _       my      _       _
baskervilles01  8       18      occupation      occupation      NN      *))))))        _       occupation
baskervilles01  8       19      .       .       .       *)      _       _       _       _
```

Figure 6.5: The sentence "Holmes was sitting with his back to me, and I had given him no sign of my occupation" annotated in the Conan Doyle corpus.

### Our System

We presented to the Shared Task a modification of the system presented in Section 6.2. The system is rule-based, we therefore needed to modify some of the rules to make it able to handle the negation structures included in the Conan Doyle corpus and the new challenges that it represents, such as a complex set of cues. The present Section also exemplifies the problems of a rule-based system when the task and the domain change.

Our system presented to the Shared Task is based on the following properties:

- It uses a static lexicon of negation cues.
- It makes use of an algorithm that traverses dependency structures (the Affected Wordforms Detection Algorithm, presented in Section 6.2.3).
- It classifies the scope of the negations by using a rule-based approach that studies linguistic clause boundaries and the outcomes of the algorithm for traversing dependency structures (the Scope Finding Algorithm, in this case a modification of the one presented in Section 6.2.3).
- It applies naive and simple solutions to the problem of classifying the negated event. The negated event, which is something that was not handled by the main system presentend in this Section (6.2), indicates the intended negated concepts. For instance, in the sentence, *No mention of that local hunt*, the word *mention* is the negated event related to the cue *No*.

- It does not use the syntactic annotation provided in the Conan Doyle corpus (just in an exception for the negated event annotation), we instead use our own dependency parser. Therefore, we participated in the open track.

Our system consists of five different modules:

- A static negation cue lexicon.
- An algorithm that from a parse given by Minipar and the negation cue lexicon produces a set of words affected by the negations.
- A rule-based system that produces the annotation of the scope of the studied sentence.
- A post-processing system that makes use of the outcomes of the initial system and produces the expected output.
- Naive rule-based approach to handle the problem of annotating the negated event.

The **lexicon** containing the negation cues is static as well as the one used in the previous system (shown in Section 6.2.3), however the old one was very small, just containing less than 20 different negation cues. Therefore, in addition to the old lexicon, we analyzed the training set and development sets provided for the Shared Task and we extracted 152 different negation cues (which were added to the ones already present in the previous system). We stored these cues in a file that feeds the system when it starts. Table 6.10 shows a small excerpt of the lexicon. The Appendix E shows the whole list of negation cues used by the system presented at the Shared Task.

| | | |
|---|---|---|
| not | no | neither..nor |
| unnecessary | unoccupied | unpleasant |
| unpractical | unsafe | unseen |
| unshaven | windless | without |

Table 6.10: Excerpt of the lexicon

We made use of the **Affected Wordforms Detection Algorithm** that uses the outcomes of Minipar. We did not modify the algorithm, therefore, it is the same described in Section 6.2.3 without modifications. It basically traverses the dependency structures and returns for each negation cue a set of words affected by the cue.

The **Scope Finding Algorithm** is the one that underwent modifications. The previous version handled the annotation as it is done in the Bioscope corpus, which is something that we had to fix in order to participate in the Shared Task. The new algorithm works as follows:

- The system opens a scope when it finds a new negation cue detected by the affected wordforms detection algorithm. In Bioscope, only the sentences in passive voice include the subject inside the scope. However, the Conan Doyle corpus does not contain this exception, including always the subject in the scope when it exists. Therefore, we modified the decision that fires this rule, and we apply the method of annotating sentences in passive voice for all the negation cues, either passive or active voice sentences.

  Therefore, for most of the negation cues the system goes backward and opens the scope when it finds the subject involved or a marker that indicates another clause, like a comma.

  There are some exceptions, such as scopes in which the cue is *without* or *neither...nor*. For them the system just opens the scope at the cue.

- The system closes a scope when there are no more wordforms to be added, i.e.:

  - It finds words that indicate another clause, such as *but* or *because*.
  - No more words in the output of the first algorithm.
  - End of the sentence.

- We also added a new rule that can handle the negation cues that are prefix or suffix of another word, such as *meaning-less*: if the system finds a cue word like this, it then annotates the suffix or prefix as the cue (such as *less*) and the rest of the word as part of the scope. Note that the Affected Wordforms Detection algorithm detects the whole word as a cue word.

In order to come up with a solution that could provide at least some results in the **negated event handling**, we came up with the following simple, rule-based system:

- When the cue word contains a negative prefix or a negative suffix, we annotate the word as the negated event.

- When the cue word is either *not* or *n't*[10] and the next word is a verb, according to the part-of-speech annotation of the Conan Doyle corpus, we annotate the verb as the negated event.

In order to produce the output as it was expected in the corpus provided in the Shared Task, described above in the Conan Doyle corpus (Morante and Daelemans, 2012), we developed a **post-processing step** that basically processes the annotated sentence with Bioscope style, (we show an example

[10]There are contractions in the Conan Doyle corpus, however, in Bioscope there are not.

for clarification: <*scope*>*There is* <*cue*>*no*</*cue*> *problem*</*scope*>) in order to produce the expected output. It tokenizes the sentences, in which each token is a word or a wordform, after that, it does the following:

- If the token contains the string <*scope*>, the system just starts a new scope column reserving three new columns and it puts the word in the first free "scope" column. Because it means that there is a new scope for the present sentence.
- If the token is between a <*cue*> annotation, the system puts it in the corresponding free "cue" column of the scope already opened.
- If the token is annotated as "negated event", the system just puts the word in the last column of the scope already opened.

Note that these three rules are not exclusive and can be fired for the same token, but in this case they are fired in the same order as they are presented.

### Results and Discussion

In this subsection we show the results obtained in two different tables: Table 6.11 shows the results of the system with the test set, Table 6.12 shows the results of the system with the development set. These tables show different evaluation metrics that are described below:

- `tp`. True positive, which means that the system predicted either the cue, the event or the scope, correctly.
- `fp`. False positive, which means that the system predicted either the cue, the event or the scope, but it is not annotated in the gold standard.
- `fn`. False negative, which means that the system did not predict either the cue, the event or the scope,, but it is annotated in the gold standard.
- Precision, Recall and F1 were also described and used in Section 6.2.4, but in this case they are counting whole scopes with the exception of the row *Scope Tokens*, in which they actually count tokens.
- The columns *gold* and *system* contain the number of appearances in the gold standard (*gold*) and the numbers of predictions (*system*).

As we can observe, the results for the development set are higher than the ones obtained for the test set. The reason is simple, we used the development set (apart from the training set) to modify the rules and to make the system able to annotate the sentences of the test set.

| **Test set** | gold | system | tp | fp | fn | precision (%) | recall (%) | F1 (%) |
|---|---|---|---|---|---|---|---|---|
| Cues: | 264 | 235 | 170 | 39 | 94 | 81.34 | 64.39 | 71.88 |
| Scopes (cue match): | 249 | 233 | 96 | 47 | 153 | 67.13 | 38.55 | 48.98 |
| Scopes (no cue match): | 249 | 233 | 96 | 48 | 152 | 66.90 | 38.96 | 49.24 |
| Scope tokens (no cue match): | 1805 | 2096 | 1222 | 874 | 583 | 58.30 | 67.70 | 62.65 |
| Negated (no cue match): | 173 | 81 | 36 | 42 | 134 | 46.15 | 21.18 | 29.03 |
| Full negation: | 264 | 235 | 29 | 39 | 235 | 42.65 | 10.98 | 17.46 |

Table 6.11: Test set results.

| **Development** | gold | system | tp | fp | fn | precision (%) | recall (%) | F1 (%) |
|---|---|---|---|---|---|---|---|---|
| Cues: | 173 | 161 | 115 | 16 | 58 | 87.79 | 66.47 | 75.66 |
| Scopes (cue match): | 168 | 160 | 70 | 17 | 98 | 80.46 | 41.67 | 54.90 |
| Scopes (no cue match): | 168 | 160 | 70 | 17 | 98 | 80.46 | 41.67 | 54.90 |
| Scope tokens (no cue match): | 1348 | 1423 | 1012 | 411 | 336 | 71.12 | 75.07 | 73.04 |
| Negated (no cue match): | 122 | 71 | 35 | 31 | 82 | 53.03 | 29.91 | 38.25 |
| Full negation: | 173 | 161 | 24 | 16 | 149 | 60.00 | 13.87 | 22.53 |

Table 6.12: Development set results.

Note that our system only detects some of the negation cues (around 72% F1 and 76% F1, respectively, for the test and development sets). We therefore believe that one of the main drawbacks of the present system is the static lexicon of cues. In the previous version (in Section 6.2) due to the simplicity of the task, the problem was not that strong. However, it is worth noting that once the negation is detected the results are competitive, we show a high precision in most of the tasks. But the recall suffers due to the coverage of the lexicon.

It is also worth noting that for the measure *Scope tokens*, which takes into account the tokens included in the scope but not a full scope match, our system provides interesting outcomes (around 63% F1 and 73% F1, respectively), showing that it is able to annotate the tokens in a similar way. We believe that this fact shows that the present system comes from a different kind of annotation and a different domain, and the extension or modification of such a system is a complex task.

We can also observe that the *negated events* results are very low (around 17.46% F1 and 22.53% F1, respectively), but this was expected because by using our two rules we are only covering two cases and moreover, these two cases are not always behaving in the same way in the corpora.

It is worth mentioning that we ranked fourth over five different systems in the *Open Track*, which is the task in which the systems could use their own resources in order to produce the annotation. Table 6.13 shows the results of all the systems just showing the precision, recall and the F1 measure. The results of the systems are shown in the table and the systems are the following:

- Uio2 (Lapponi et al., 2012), it is a data-driven approach based on semantic and syntactic dependency features.
- UGroningen (r1 and r2) (Basile et al., 2012), it has two different runnings and both are based in discourse structures.
- UCM-1 (Carrillo de Albornoz et al., 2012), it is a rule-based approach based on phrase structure parsing.
- UCM-2 (Ballesteros et al., 2012), it is our system presented in this Section.

| System | Cues | | | Scopes (cue match) | | | Scopes (no cue match) | | | Scope Tokens | | | Negated | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Prec. | Rec. | F1. | Prec. | Rec. | F1. | Prec. | Rec. | F1. | Prec. | Rec. | F1. | Prec. | Rec. | F1. |
| UiO2 | 89.17 | 93.56 | **91.31** | 85.71 | 62.65 | **72.39** | 85.71 | 62.65 | **72.39** | 82.25 | 82.16 | **82.20** | 66.90 | 57.90 | **61.79** |
| UGroningen r2 | 88.89 | 84.85 | 86.82 | 76.12 | 40.96 | 53.26 | 76.12 | 40.96 | 53.26 | 69.20 | 82.27 | 75.17 | 56.63 | 65.29 | 60.65 |
| UCM-1 | 89.26 | 91.29 | 90.26 | 82.86 | 46.59 | 59.64 | 82.86 | 46.59 | 59.64 | 85.37 | 68.53 | 76.03 | 66.67 | 12.72 | 21.36 |
| **UCM-2 (ours)** | 81.34 | 64.39 | 71.88 | 67.13 | 38.55 | 48.98 | 66.90 | 38.96 | 49.24 | 58.30 | 67.70 | 62.65 | 46.15 | 21.18 | 29.03 |
| UGroningen r1 | 86.90 | 82.95 | 84.88 | 46.38 | 12.85 | 20.12 | 46.38 | 12.85 | 20.12 | 69.69 | 70.39 | 69.99 | 53.94 | 52.05 | 52.98 |

Table 6.13: Results of the Systems presented in the Shared Task.

As noted above, we certainly could observe that our system is competitive in the scope annotation but it suffers due to the static lexicon of cues that produces a substantial drop in the final accuracy. The recall for the cues drops down to the 64.39%, however and in spite of that, our scope annotation recall is more or less in the same range as the other systems, with the exception of UiO2 which provides much better results. We believe that our system in synergy with an accurate cue detector could provide much better results. It would be interesting to try this in order to find how far it can go with a lexicon that could cover more negation cues that appear in the test sets, as happened in the case of Bioscope, in which we covered more than 90% of the cues in the three studied cases (see Table 6.7).

Note that all the systems were developed for the Shared Task, however, our system was a modification of the one prepared to handle the Bioscope annotation, which is presented in this Section. This fact evidences that modifying a rule-based approach in order to handle a new way of annotating sentences (in this case very different and more complicated) is a daunting task and it could provide interesting results but only after a lot of iterations.

We here analyze the different errors of our system with respect to the development set. This set contains 787 sentences, 144 of them were sentences containing negations, with 168 scopes, 173 negation cues and 122 negated events.

Concerning the negation cue detection we obtained 58 false negatives (fn) and 16 false positives (fp). These results were not only derived from the static lexicon of cues. The main problem was related with the management of sentences with more than one scope. The majority of the errors were

produced because in some cases all the cues are assigned to all the scopes detected in the same sentence, generating false positives, and in other cases the cues of the second and subsequent scopes were ignored, generating false negatives. The first case occurred in sentences like (1), *no* and *without* were labeled as cues in the two scopes. The second case occurs in sentences like (2), where neither the second scope nor the second cue were labeled. In sentence (3) *un* is labeled as a cue twice (*unbrushed, unshaven*) but within the same scope, generating a false positive in the first scope and a false negative in the second one.

- (1) But no [one can glance at your toilet and attire without [seeing that your disturbance dates from the moment of your waking..']].
- (2) [You do ]n't [mean] - . [you do] n't [mean that I am suspected]?".
- (3) Our client smoothed down [his] un[brushed hair] and felt [his] un[shaven chin].

We also found false negatives that occur in multi word negation cues as *by no means*, *no more* and *rather than*.

A different kind of false positives is related to modality cues, dialogue elements and special cases (Morante and Blanco, 2012). For example, *no* in (4), *not* in (5) and *save* in (6).

- (4) "You traced him through the telegram, no [doubt]." said Holmes.
- (5) "All you desire is a plain statement, [is it] not?'.
- (6) Telegraphic inquiries... that [Marx knew] nothing [of his customer save that he was a good payer].

We can also find problems with affixal negations, consisting in a not correct separation of the affix and the root of the word. For example, in (7) *dissatisfied* was erroneously divided in *di-* and *ssatisfied*. Again, it is derived from the use of a static lexicon of cues, which is not able to handle all the possible appearances.

- (7) He said little about the case, but from that little we gathered that [he also was not dis[satisfied] at the course of events].

Finally, we could also find cases that may be due to annotation errors. For example, *incredible*, in (8), is not annotated as negation cue. We think that the annotation of this cue is not consistent, it appears five times in the training corpus, twice is labeled as cue, but three times is not. According to the context in this sentence, *incredible* means not *credible*.

- (8) "Have just had most incredible and grotesque experience.

Focusing on the full scope detection, most of the problems were due again to the management of sentences with more than one scope. We have obtained 98 false negatives and 17 false positives. Most of the problems were related with affixal negations, as in (9), in which all the words were included in the scope, which according to the gold standard is incorrect.

- (9) [Our client looked down with a rueful face at his own] un[conventional appearance].

In the scope tokens annotation, the results were better, reaching 73% F1 in scope tokens compared to 55% in full match scopes. It is worth noting that annotating the whole scope is a much more complicated task than annotating just some tokens.[11] The reason was mainly that our system included tokens for the majority of scopes, increasing the recall until 75% but lowering the precision due to the inclusion of more false positives.

#### Conclusions about the Participation in the Shared Task

As the main conclusion we can say that modifying such a system to perform in a different type of texts and a different level of annotation is complicated. However, taking into account this fact, and the results obtained, we are tempted to say that our system presents competitive results.

We believe that the present system can be improved in different ways:

- Improve the management of sentences with more than one scope modifying the scope classification algorithm and the post-processing step.
- Replacing the dependency parser with a state-of-the-art parser in order to get higher performance.
- Proposing a different way of getting a more competitive lexicon of cues, by using a semantic approach that informs if the word has a negative meaning in the context of the sentence.

## 6.3 Chapter Summary

In this Chapter we have shown two application works in which we made use of dependency structures. We studied text simplification and the inferring of the scope of negation cues, providing two rule-based approaches that are mainly-based on traversing dependency structures.

[11] In the same way, in Section 6.2 when we used the PCS measure we calculated the whole scope match recall.

# Chapter 7

# Conclusions and Future Work

*Finally, in conclusion, let me say just this.*
*Peter Sellers.*

This last Chapter summarizes the main conclusions and suggestions for future work of the present thesis. In this thesis we studied dependency parsing from several perspectives:

1. Studying the problem focusing on the corpora size and sentence length.
2. Improving the outcomes of the parsers.
3. Optimizing the parsers or so called, machine learning models.
4. Applying dependency structures to solve NLP problems.

The first three were based on data-driven parsers and in the last one we used a rule-based parser. Our initial idea was to close the circle by providing contributions in several aspects of the dependency parsing problem, we believe that we have reached our initial objectives.

As we introduced in Section 1.2 the aim of this thesis was to answer the following questions:

- *Which factors in the training corpus can affect the accuracy in the analysis phase?*
- *Is it possible to improve accuracy by manipulating the training corpus?*
- *Is it possible to improve accuracy by combining different parsers that are produced by the same parser generator?*

- *Is it possible to modify the behavior of transition-based parsing algorithms with the intention of improving the performance?*
- *Is it possible to modify the behavior of graph-based parsing algorithms with the intention of improving the performance?*
- *Is it possible to automate the optimization of dependency parsers?*
- *Is it possible to create an automatic (and accurate) feature selection system for a transition-based parser? or even, a machine-learning based parser?*
- *Can we use dependency structures to solve NLP problems?*

In the next Sections, we provide an answer to all of them in turn.

## 7.1 Conclusions about Analyzing Dependency Analysis

In this section we summarize our answers to the questions shown below looking into the results and studies of Chapter 3 entitled *Analyzing Dependency Analysis*:

### 7.1.1 Which factors in the training corpus can affect the accuracy in the analysis phase?

There are various factors in the training corpus that are very important for the purpose of parsing and training parsing models, such as, sentence length or training corpora size. In Section 3.1 and especially in Section 3.3 we demonstrated that not always having more words means better results, showing as a conclusion that the quality of the sentences, and even more importantly the annotation, deeply affects the final accuracy. Moreover, we have demonstrated that longer sentences (in Section 3.2) are very rich in training time for a transition-based parser, because they contain syntactic structures that lead the parser to learn several transitions. We have also provided evidences that long sentences affect the final accuracy because the parsers have problems parsing them, the attachment scores and complete-match results drop as soon as the corpus contain long sentences, as we saw in Section 3.4. We have suggested for future corpora developments to think carefully about the sentences that are going to be annotated, because it seems that sentences that share the same syntactic structure of other sentences included in the data set do not provide an improvement in the final accuracy.

Moreover, in Section 3.4 we have evaluated a big set of data-driven parsers in a different way focusing on complete-match accuracy (or sentence-based evaluation measures), evidencing that modifying the point of view

shows up problems that were not seen as a first sight, such as the importance of sentence length and some parser issues.

### 7.1.2 Is it possible to improve accuracy by manipulating the training corpus?

We certainly can say that manipulating the training corpus we can alter the final accuracy in a very significant way. In Section 3.1 we saw that the accuracy is homogeneous, nevertheless, it seems very smart to select the sentences that are going to be used carefully, because there could be significant differences. A training corpus rich with long sentences is therefore full of different syntactic structures and we can expect better results.

## 7.2 Conclusions about Enhancing Dependency Analysis

In this section we provide our answers to the questions shown below looking into the results and studies of Chapter 4 entitled *Enhancing Dependency Analysis*:

### 7.2.1 Is it possible to improve accuracy by combining different parsers that are produced by the same parser generator?

This question is related with the feasibility study shown in Section 4.1, in which we studied how to train some specific parsers in order to parse some function words in which the parser frequently fails. We have demonstrated that the problem is feasible. The study shows that is interesting to face the problem of enhancing the accuracy focusing on a small part of the sentences providing significant results because the selected words (prepositions, nexus and conjunction) are very frequent and they are very important because they act as connectors in the sentences and therefore in the dependency structures.

### 7.2.2 Is it possible to modify the behavior of parsing algorithms with the intention of improving the performance?

Actually, this question can be divided in two different questions:

- *Is it possible to modify the behavior of transition-based parsing algorithms with the intention of improving the performance?*
- *Is it possible to modify the behavior of graph-based parsing algorithms with the intention of improving the performance?*

These two questions concern the work in which we studied the root position during parsing and training time, shown in Section 4.2. We believe that we have provided very interesting conclusions showing that the root position is indeed relevant, and very important for some widely used parsing algorithms in which the left attachments are greedily generated, such as the Nivre arc-eager (See Section 2.1.1). Therefore, the answer to these questions is basically yes. We believe that our findings show that the root position must be taken into account in future transition-based parsing developments. However, it is also true, that changing the root position, or forcing the parser to work with no root at all, in some transition-based parsing algorithms (in which the left attachments are not greedily generated) or graph-based parsers (such as MSTParser models) does not produce important differences, providing just noise between one scenario to another. In the study we focused on the Nivre arc-eager and the Nivre arc-standard parsing algorithms because they basically conform the two different parsing orders that a transition-based parsing algorithm can have (see Figure 5.2 and Figure 5.3 and the accompanying discussion in Chapter 5).

As a main conclusion we can say that there is a lot of room for improvement. We have demonstrated it during this thesis, showing improvements focusing on two very different problems inherently related with dependency parsing.

## 7.3 Conclusions about Optimizing Dependency Analysis

In this section we provide our answer to the question shown below looking into the results and studies of Chapter 5 entitled *Optimizing Dependency Analysis*:

### 7.3.1 Is it possible to automate the optimization of dependency parsers?

Showing the outcomes of MaltOptimizer, which is the system presented in Chapter 5, a direct answer to the questions shown above is simply yes. We have demonstrated that automating the process of optimizing MaltParser focusing on preliminary parameters, parsing algorithm and more important automatic feature selection is feasible. We have shown that it is even possible to beat the results provided by manual optimization, even manual optimization carried out by experienced researchers, because the search in the feature space and the different combinations make an intended manual exhaustive search impractical.

MaltOptimizer is an expert system, that also provided us a lot of knowledge about how a transition-based parser generator works, and the rather

different insights that it represents. We certainly expect that our published system will be used by the dependency parsing community during the next years due to the importance of MaltParser in the community and due to the significant results that we provided with it.

We have also demonstrated that with this system is possible to optimize a single parsing algorithm (see Appendix D), stopping the process between phases and running the feature selection experiment for a desired parsing algorithm, providing substantial results for each of the windows in which we can add and remove features to and from the default feature models.

### 7.3.2 Is it possible to create an automatic (and accurate) feature selection system for a transition-based parser? or even, a machine-learning based parser?

Inside MaltOptimizer, there is an algorithm capable of selecting the optimal features for a given training set and the corresponding parsing algorithm. Therefore, the answer to this question is again yes, we have demonstrated that automatic feature selection for transition-based dependency parsing is possible and we demonstrated that it is possible to provide very good outcomes.

## 7.4 Conclusions about Applying Dependency Analysis

In this section we provide our answer to the question shown below looking into the results and studies of Chapter 6 entitled *Applying Dependency Analysis*:

### 7.4.1 Can we use dependency structures to solve NLP problems?

From the outcomes of the systems presented in Chapter 6, the answer is yes. We have demonstrated that dependency structures can be very useful to solve different problems related with natural language processing.

In Section 6.1 we have demonstrated that they are useful to simplify sentences that have been previously parsed ensuring the grammatical correction of the final outcome. Moreover, in Section 6.2, we show that exploring dependency structures is possible in order to find how far the action of negation cues goes in a given sentence. The results of both implemented approaches show that rule-based approaches that make use of dependency structures are something feasible.

However, it is worth noting that rule-based systems, as the one presented in Chapter 6, have an important issue, basically that modifying such a system in order to change the domain, the corpus or the task is complicated.

This fact is evidenced in Section 6.2.6 in which we show how we updated the system presented in Section 6.2 in order to handle the negations from a different corpus. This is why, we suggest that dependency structures can be used as features in machine learning approaches that solve the same problem.

As a final conclusion we can emphasize that we are glad of prove that dependency parsing can be a very useful tool and it can be used in order to perform different application works.

## 7.5 Future Work

In the near future, we would like to keep working on these topics but in a rather different way. We would like to apply what we have learned during the elaboration of this thesis.

One of the issues that motivates us more is to apply the knowledge acquired in dependency parsing to different topics, such as extracting information from social media text. We consider that a good idea could be the application of new parsers capable of annotating semantic and syntactic information and apply, in a new way, some of the ideas that we tested during this thesis in which we were able to extract the negated information from texts, but this is just an example. We believe that the acquired experience in feature selection, optimization and machine learning in general could provide an interesting way of carrying out these new experiments.

It is also worth noting that it might be interesting to research and implement robust and universal parsers without annotating several corpora for several languages, instead having a universal parser that is capable of providing competitive results for every language. The first step should be a universal guideline about the annotation of sentences in the corpora. This work has already been started by Petrov, Das and McDonald (2012).

We show during the next subsections specific research directions for each of the perspectives that we studied in the present thesis.

### 7.5.1 Analyzing Dependency Analysis

It would be interesting to perform similar experiments as the ones that we carried out and explained in Chapter 3, with different corpora and/or parsers. However, it is true that this part of the thesis served us to learn the topic and comprehend the behavior of the parsers. Nevertheless, it is also true that we certainly will include similar studies as the ones that we presented in Chapter 3 in future parsing developments in order to evaluate the parsers and/or algorithms.

### 7.5.2 Enhancing Dependency Analysis

The section of this thesis concerned with enhancing parsing accuracy is the one that has more room for improvements. The best dependency parser models achieve nowadays around 80% - 90% LAS, therefore there are more than 10 percentage points of improvement in all the possible scenarios. We certainly believe that we could come up with new algorithms applying new programming techniques. As soon as the computers become bigger and faster we would also be able to run the experiments quickly and we will be able to produce better parsing models.

Concerning the parsing combination problem, there could be new ways of finding output (or input) combinations by exploring ways of improving the accuracy. However, it is also true that there is a lot of research concerning this topic as we saw in Section 2.3, so it seems quite unlikely that we (or other researchers) could come up with hybrid systems that perform in a better way.

Another idea that we would like to explore is the study, development, and modification of current transition-based and graph-based dependency parsing algorithms. We certainly believe that there is a lot of room for improvement in the current state-of-the-art, both for the final performance of such a system and the training and testing time. The needs that a parser has, considering time, memory and disk space issues, are a bottle-neck that should be addressed in the future. This fact is even more interesting if we want to apply these parsers to small computers, such as mobile phones or tablet pc's, applying them, for instance, over texts extracted from social media in an online way. However, it is true that most parsers are quite fast in testing time, but when we want to apply an online[1] learning algorithm this fact may be an issue, which we would like to study and face in the future.

### 7.5.3 Optimizing Dependency Analysis

We would like to explore different automatic feature selection algorithms in a similar way as we started in Appendix C. This may involve the development of a more advanced optimization strategy that interleaves the optimization of parsing algorithm, feature model and learning algorithm instead of using a greedy stepwise approach in order to avoid the highly correlation between the algorithms, features and/or parameters.

Besides that, we consider that the optimization branch of the thesis could be extended and applied to different dependency parsers that use a rich feature set specification. In the case of transition-based parsers it could be handle simply transforming the produced feature set for the new feature

[1] An online machine learning algorithm is the one that is able to update the model any time considering the outcomes provided by, for instance, parsing new sentences.

specification language. In other kind of parsers, for instance, maximum-spanning-tree parsers, the features are simple because we do not have to worry about the data structures that are present in transition-based parsing but we could consider the same kind of features: part-of-speech, morphology, etc. Moreover, it is worth remarking that our methods can be applied to all MaltParser algorithms, we therefore believe that it could be a universal way of tuning the features of transition-based parsers because MaltParser actually includes different ways of solving the parsing problem and we are able to select between them and modify all of them.

Finally, our acquired experience in automatic feature selection might be interesting in a cross-domain and cross-lingual experimental set up, in which we would like to study how it is possible to apply the automatic feature selection techniques to an online learning algorithm, not only updating the parser behavior, but updating the features that are relevant for the data set that would be being used for training.

### 7.5.4 Applying Dependency Analysis

Concerning the text simplification system that we presented in this thesis, we certainly can improve the outcomes of the system very deeply. The system presented here is just a first step towards a complex rule-based text simplification system. However, it is good that the produced sentences are grammatically correct and we emphasize that this is due to the potential usefulness that dependency parsing provide. Therefore, there could be new rules, hybrid combinations applying morphologic or semantic simplification.

Concerning the system that is able to infer the scope of negation cues it is very likely that we would be able to improve the outcomes by using a parser that may produce results close to the ones already published in the state of the art, such as MaltParser. However, it is true that by using Minipar we did not need to annotate the sentences because it is a rule-based parser and the outcomes were satisfactory. Another idea for future work would be to include different kind of parsers, by applying the following ideas:

- By using one of the systems of the CoNLL 2009 Shared Task (Hajič et al., 2009), such as (Bohnet, 2009), and using the semantic annotation provided.

- By using partial parsing or clause segmentation (Carreras, Màrquez, and Castro, 2005) or one of the systems of the CoNLL 2001 Shared Task (Tjong Kim Sang and Déjean, 2001).

We certainly believe that these technologies could provide very useful information in order to infer the scope of negation.

## 7.6 Chapter Summary

In this last Chapter of the manuscript we have shown the main conclusions and suggestions for future work, taking our recent acquired experience into account. We have summarized all the questions introduced in the first chapter and we have provided an answer for each of them by commenting our own work which is shown in all the previous chapters.

Finally, we have shown some suggestions for future work studying what we would like to do in the future, showing future parsing developments and future tasks and showing that our experience could provide us a very good path in order to face all of them.

# Part II

# Resumen de la Tesis en español: Análisis, Optimización, Mejora y Aplicación del Análisis de Dependencias.

# Análisis, Optimización, Mejora y Aplicación del Análisis de Dependencias


**Tesis Doctoral**

Presentada por
**Miguel Ballesteros Martínez**

Bajo la dirección de los Doctores

**Dra. Virginia Francisco Gilmartín**

**Dr. Pablo Gervás Gómez–Navarro**

Facultad de Informática
Universidad Complutense de Madrid
Madrid 2012

# Chapter 8

# Introducción

El rendimiento de los analizadores de lenguaje estadísticos ha mejorado tremendamente durante las dos últimas decadas, y ahora hay un gran número de sistemas que pueden usarse para generar analizadores para nuevos idiomas y nuevas aplicaciones. Esto incluye analizadores de constituyentes como el Analizador de Stanford (Klein and Manning, 2002) o el Analizador de Berkeley (Petrov et al., 2006), y también incluye analizadores de dependencias como MaltParser (Nivre, Hall, and Nilsson, 2004) y MSTParser (McDonald, Crammer, and Pereira, 2005).

La comunidad de análisis de dependencias es importante dentro de la comunidad de Procesamiento de Lenguaje Natural (PLN), y la comunidad de lingüística computacional (CL). Se puede observar como en las conferencias recientes e importantes, incluso en el presente año, hay diferentes sesiones dedicadas al análisis de dependencias en los que los investigadores han tenido, y tienen, la oportunidad de mostrar sus contribuciones. Un ejemplo claro, puede encontrarse en las Tareas competitivas de la décima y la undécima Conferencia de Aprendizaje Computacional del Lenguage Natural (CoNLL en sus siglas en inglés, Conference of Computational Natural Language Learning). Estas tareas competitivas estaban completamente centradas en Análisis de Dependencias multilingüe (Buchholz and Marsi, 2006; Nivre et al., 2007). Se utilizó un conjunto de diferentes idiomas y se estudió el análisis desde diferentes perspectivas. Otro ejemplo, puede ser la reciente conferencia del European Chapter for Computational Linguistics of 2012 (EACL) (Daelemans, Lapata, and Màrquez, 2012), donde el premio al mejor artículo fue concedido a un trabajo sobre análisis de dependencias (Luque et al., 2012), y hubo multitud de trabajos interesantes relacionados, como pueden ser (Bohnet and Kuhn, 2012), (Farkas, Vincze, and Schmid, 2012), (Tsarfaty, Nivre, and Andersson, 2012), (Gómez-Rodríguez and Fernández-González, 2012) y (Ballesteros and Nivre, 2012).

Creemos que la comunidad del Análisis de Dependencias está teniendo mucho interés porque hay todavía mucha capacidad de mejora, y existe la

posibilidad de encontrar diferentes formas de resolver mejor los problemas relacionados, incorporando el desarrollo de nuevas ideas.

El área de aplicacion del análisis de dependencias está en constante crecimiento, y dada la trayectoria, en el futuro, habrá diferentes contribuciones donde los investigadores harán uso de las mejoras en análisis de dependencias, para resolver diferentes tareas del procesamiento del lenguaje natural. Por ejemplo, se pueden ya encontrar diferentes aplicaciones donde se usa el análisis de dependencias como principal herramienta de trabajo, como reconocimiento de implicación textual (Herrera, Peñas, and Verdejo, 2005), extracción de relaciones (Culotta and Sorensen, 2004), recursos para la traducción automática (Tiedemann, 2012), sistemas de pregunta-respuesta (Cui et al., 2005), detección de negaciones (Councill, McDonald, and Velikovich, 2010), generación de sinónimos (Shinyama, Sekine, and Sudo, 2002), realización superficial (Bohnet et al., 2011), incremento de recursos léxicos (Snow, Jurafsky, and Ng., 2005), análisis del contenido procedente de redes sociales (Foster et al., 2011) o simplificación de textos mediante la implementación de un algoritmo que poda árboles de dependencias (Ballesteros, Bautista, and Gervás, 2010).

En esta tesis tuvimos la idea original y la oportunidad de estudiar el análisis de dependencias en profundidad, aplicando los analizadores a diferentes dominios, temas e idiomas, adquiriendo, de ese modo, conocimiento experto que podría ser muy útil para el futuro. Por ello, hemos estudiado el problema del análisis de dependencias desde cuatro perspectivas distintas:

1. ¿Cómo podemos analizar el problema?

2. ¿Se puede mejorar la precisión modificando el funcionamiento y el flujo de análisis de los analizadores?

3. ¿Se puede optimizar automáticamente un generador de analizadores de dependencias?

4. ¿Se puede aplicar el análisis de dependencias para resolver problemas inherentes al procesamiento del lenguaje natural?

Todo el trabajo incluido en esta tesis está intrínsicamente conectado, ya que es practicamente imposible resolver uno de los problemas mencionados sin las nociones adquiridas en los demás. En el presente Capítulo, se describe el problema de las dependencias en la teoría lingüística y el análisis de dependencias de manera formal. Además, se muestran los objetivos y la estructura de esta tesis.

## 8.1 Lingüística de Dependencias y Análisis de Dependencias

La teoria moderna de dependencias en el lenguaje viene de Lucien Tesnière (1959) con el objetivo de generar una gramática útil para la enseñanza de idiomas, sin embargo, otros investigadores dijeron que el concepto de dependencia en el lenguaje fue utilizado durante la Edad Media, como es el caso de (Covington, 1984). Hubo otros en los sesenta y los setenta que enfatizaron la idea de dependencias en la lengua, como Hays (1964) o Robinson (1970). Además, la teoría de dependencias de Igor Mel'čuk en los ochenta es muy relevante (1988), porque estableció las bases de lo que hoy en día entendemos por dependencias en el lenguaje, en el contexto de la meaning text theory (MTT), especialmente para idiomas eslavos ya que permiten mayor 'libertad' en el orden de las palabras. Durante los últimos años y a pesar de la gran tradición en lingüística descriptiva, el análisis de dependencias está adquiriendo y teniendo un gran interés en la comunidad de lingüística computacional, como ejemplo, Nivre (2005) argumentó que el reciente boom que está teniendo las gramáticas basadas en dependencias es debido a la gran utilidad que aportan las relaciones de dependencias entre palabras.

La idea básica es que la estructura sintáctica en el lenguaje consiste en diferentes elementos léxicos unidos mediante relaciones asimétricas llamadas dependencias. Una estructura de dependencias es, por tanto, un grafo etiquetado y dirigido, que consiste en un conjunto de nodos, etiquetados con palabras, y un conjunto de arcos dirigidos que pueden estar etiquetados, o no, con tipos de dependencias. El análisis de dependencias es principalmente el análisis sintáctico del lenguaje natural, basado en relaciones de dependencias sintácticas. Teniendo esto en cuenta, y teniendo un analizador de dependencias, somos, por lo tanto, capaces de construir una estructura de dependencias dada una frase ejemplo (Nivre, 2006; Kübler, McDonald, and Nivre, 2009).

### 8.1.1 Estructuras de Dependencias

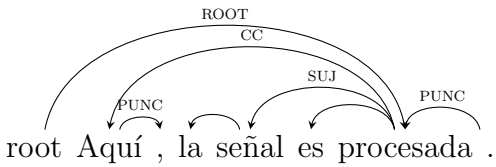
De manera formal, un **grafo de dependencias** para una frase
S = $w_1$,...,$w_n$ es un grafo dirgido G = (V,A), donde:

- V = $\{1, ....n\}$ es el conjunto de **nodos**, representando las palabras o formas de palabra.
- A $\subseteq$ V x V es el conjunto de **arcos**, representando las dependencias.
- Un arco i $\rightarrow$ j es una dependencia con padre $w_i$ e hijo $w_j$.
- Un arco i $\rightarrow$ j puede estar etiquetada con un tipo de dependencia contenido en el conjunto de posibles dependencias.

Además, un grafo de dependencias debe respetar las siguientes limitaciones:

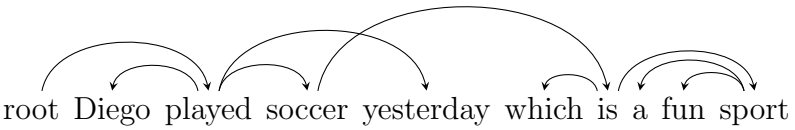
- El grafo debe ser conexo y acíclico.
- Un nodo puede ser destino de más de un arco. Algunas limitaciones más simples, requieren que un nodo sólo puede ser destino de un único arco.

En este manuscrito nos referiremos a las estructuras de dependencias como **árboles de dependencias** (o incluso árboles), por razones de simplificación. Además, en algunos casos[1] se permite que sean *grafos* pero en la mayoría de los propósitos de esta tesis, las estructuras de dependencias son realmente árboles de dependencias. La figura mostrada abajo es un ejemplo de un árbol de dependencias etiquetado para una frase escrita en castellano.

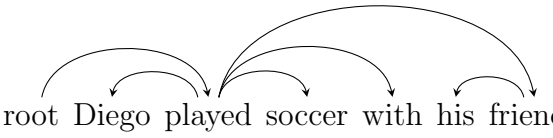



Los árboles de dependencias pueden ser **proyectivos** o **no proyectivos**. Mostramos aquí una definición formal de proyectividad: un árbol de dependencias es proyectivo si, y sólo si, para cada arco wi→wj, y cada nodo wk entre wi y wj en el orden original, hay un camino directo entre wi y wk.

El siguiente árbol de dependencias es, por lo tanto, no proyectivo, porque no hay un camino directo entre *soccer* a *yesterday* en el orden original de las palabras:




y el siguiente árbol es, por lo tanto, proyectivo, porque hay un camino directo entre dos nodos cualesquiera contenidos en el árbol:




[1] El nodo raíz, que es el que permite tener árboles de dependencias forzando una única raíz para cada frase.

### 8.1.2 Análisis de Dependencias

El problema del análisis sintáctico de dependencias consiste en construir un sistema que es capaz de generar un árbol de dependencias dada una frase de entrada. Si tenemos una frase de entrada $w_1 \dots w_n$, el objetivo de un analizador de dependencias es asignarle un *grafo de dependencias*, el cual es un grafo dirigido $G = (V, A)$ donde $V = \{1, \dots, n\}$ y $A \subseteq V \times V$.[2] Normalmente, asumimos que los grafos (o árboles) deben ser acíclicos y un único padre por cada nodo (que un nodo sea destino de más de un arco).

De manera formal, podemos decir que un analizador de dependencias maximiza la puntuación internamente en el sistema de aprendizaje automático de un grafo producido analizando una frase dada, como se muestra abajo:

- Entrada: S = $w_1$,...,$w_n$
- Salida: G* = argmax F(S,G) donde G∈G(S)
  - F(S,G) = puntuación de G para S
  - G(S) = espacio de búsqueda para S

Hay diferentes tipos de sistemas que resuelven el problema del análisis de dependencias. Además de los antiguos analizadores basados en reglas que están basados en gramáticas y un complejo conjunto de reglas. Existen principalmente dos familias que están basadas en **aprendizaje automático**:

- Los **Graph–based model** (o modelos basados en grafos) (Eisner, 1996), parametrizan los modelos de análisis mediante arcos de dependencias e intentan predecir los árboles completos dando (y clasificando) peso a todos los posibles arcos para la frase dada. El ejemplo más representativo es **MSTParser** (McDonald et al., 2005; McDonald, Lerman, and Pereira, 2006), pero podemos encontrar otros sistemas como el de Corton (2006), el de Dreyer (2006) o los de Carreras (2007; 2008).

- El modelo basado en **transiciones** (Nivre, Hall, and Nilsson, 2004) (Yamada and Matsumoto, 2003), donde los modelos de análisis son parametrizados por transiciones de estado (o máquinas de estados) cuyas acciones (o transiciones) manipulan las palabras de entrada y construyen las relaciones de dependencias entre ellas. **MaltParser** (Nivre, Hall, and Nilsson, 2006) el el sistema que mejor representa esta categoría. Pero además existen otros, como el de Johansson (2006), el de Cheng (2006) o el de Wu (2006).

[2] En practica, los arcos del conjunto $A$ pueden estar etiquetados, pero ignoramos las etiquetas de los arcos en esta explicación por simplificación de la presentación.

Son básicamente **generadores de analizadores**, porque devuelven un modelo que es capaz de analizar frases usando un corpus de datos anotados (o **corpus de entrenamiento**) para el entrenamiento. Por eso, se les llama, basados en aprendizaje automático, o basados en datos, ya que replican el comportamiento aprendido cuando tienen que anotar nuevas frases. Ambas tecnologías fueron establecidas como dominantes durante las tareas competitivas de análisis de dependencias de las conferencias CoNLL(Buchholz and Marsi, 2006; Nivre et al., 2007).

### Formato de Datos y Atributos

El formato de datos de las tareas competitivas de las conferencias CoNLL, contiene los siguientes atributos que son asumidos por los analizadores de dependencias como formato de entrada y salida:

- Estos son los atributos usados como entrada:
  1. FORM: Forma de palabra. Es básicamente la palabra o la forma de palabra.
  2. LEMMA: Lema. Es la forma canónica de la palabra, por ejemplo, *hablar, hablas, hablan, hablado* and *hablando* vienen del mismo lexema, siendo *hablar* su lema.
  3. CPOSTAG: Categoría gramatical de grano grueso.
  4. POSTAG: Categoría gramatical de grano fino. La diferencia entre CPOSTAG y POSTAG es básicamente que CPOSTAG es menos específico, por ejemplo, para la palabra *puente*, el CPOSTAG podría ser nombre, pero el POSTAG podría ser nombre común.
  5. FEATS: Lista de atributos morfosintácticos (por ejemplo, caso, número, genero, tiempo verbal, etc.). Los atributos están normalmente separados por una barra vertical '|'.
- Estos son los atributos que deben ser producidos por los analizadores, sin embargo, los últimos dos sólo se producen en algunos casos:
  1. HEAD: Identificador del padre de la palabra.
  2. DEPREL: Relación de dependencia con el HEAD (o padre).
  3. PHEAD: Identificador del padre, pero proyectivo. Esta columna garantiza una estructura de dependencias proyectiva, a diferencia de la columna HEAD. No está disponible en la mayoría de los corpora teniendo normalmente una barra baja que indica no disponibilidad en las últimas dos columnas (PHEAD y PDEPREL).
  4. PDEPREL: Relación proyectiva con el head. Es la relación proyectiva con PHEAD si está disponible.

En la Figura 8.1 mostramos un ejemplo de una frase anotada en formato CoNLL, donde cada columna muestra la información mostrada arriba siguiendo el mismo orden.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | No | no | r | rn | _ | 3 | NEG | 3 | NEG | | |
| 2 | habrían | haber | v | va | num=p\|per=3\|mod=i\|tmp=c | 3 | _ | _ | 3 | _ |
| 3 | pasado | pasar | v | vm | gen=m\|num=s\|mod=p | 0 | ROOT | 0 | ROOT | |
| 4 | más_de | más_de | r | rg | _ | 5 | _ | 5 | _ | |
| 5 | seis | seis | d | dn | num=p\|gen=c | 6 | _ | 6 | _ | |
| 6 | meses | mes | n | nc | gen=m\|num=p | 3 | SUJ | 3 | SUJ | |
| 7 | desde | desde | s | sp | for=s | 3 | CC | 3 | CC | |
| 8 | su | su | d | dp | num=s\|per=3\|gen=c | 9 | _ | 9 | _ | |
| 9 | boda | boda | n | nc | num=s\|gen=f | 7 | _ | 7 | _ | |
| 10 | . | . | F | Fp | _ | 3 | PUNC | 3 | PUNC | |

Figure 8.1: La frase escrita en castellano, "No habrían pasado más de seis meses desde su boda", anotada en formato CoNLL.

El corpus de entrenamiento contiene toda la información para cada palabra, incluidos los atributos de salida,[3] teniendo el analizador toda la información para realizar el aprendizaje. Sin embargo, los **corpus de test** (los ciegos, que sirven para la evaluación) deben tener los atributos de salida ocultos, o simplemente, no disponibles.

Los analizadores basados en aprendizaje automático utilizan **features (o atributos)** para definir qué atributos del formato de datos son útiles construyendo un **modelo de features** que es un subconjunto del espacio de features, sin embargo cada analizador lo hace de manera distinta. Los features están normalmente basados en la lista anterior, incluida la columna DEPREL, pero no la columna HEAD. Con lo que los features pueden ser principalmente, categorías gramaticales, el lema de las palabras, las palabras por si mismas, información morfológica como el género o el número, pero también pueden ser features basados en las estructuras de dependencias parcialmente construidas (DEPREL column).

Los atributos LEMMA y FEATS no están siempre disponibles en todos los corpora, además CPOSTAG y POSTAG pueden ser idénticos. Es importante tener en cuenta que los atributos DEPREL sólo están disponibles dinámicamente en la estructura de dependencias parcialmente construida.

Los modelos basados en transiciones (o transition-based) usan los features para decidir qué transiciones son más probables y disparar las diferentes acciones del autómata, sin embargo los modelos basados en grafos principalmente usan los features para dar peso a los posibles arcos entre palabras y poder seleccionar finalmente los arcos más valorados.

[3]En algunos casos, las columnas (o atributos) pueden no estar disponibles.

### Evaluación

Existen diversas maneras de evaluar los analizadores de dependencias con el objetivo de obtener la precisión, normalmente tenemos un corpus de entrenamiento (para cada idioma) que se utiliza para el entrenamiento, es decir, dejar al analizador aprender sobre cómo debe anotar las frases de test. También tenemos un corpus de test (que contiene toda la información, pero no los corpus de test ciegos que no contienen los atributos de salida) que son normalmente pequeños y sirven como estándar para comparar entre analizadores. Las siguientes medidas de evaluación son las que se usan en esta tesis y las publicaciones del estado del arte.

- **LA**: Label Accuracy. Mide el porcentaje de palabras evaluables[4] que ha sido correctamente anotado con las etiquetas de dependencia.
- **UAS**: Unlabeled Attachment Score. Mide el porcentaje de palabras evaluables para los que el sistema ha predicho el padre de manera correcta.
- **LAS**: Labeled Attachment Score. Mide el porcentaje de palabras evaluables para los que el sistema ha predicho el padre y la etiqueta de dependencias correctamente.
- **UCM**: Unlabeled Complete–Match. Mide el porcentaje de frases para los que el sistema ha predicho de manera correcta todo el árbol (no etiquetado) de dependencias.
- **LCM**: Labeled Complete–Match. Mide el porcentaje de frases para los que el sistema ha predicho de manera correcta todo el árbol de dependencias.

La medida de evaluación más común es LAS (o labeled attachment score), por ello en la mayoría de los casos es la que se utiliza. Sin embargo, en otros casos, merece la pena buscar otro tipo de información utlizando una medida de evaluación diferente. También existen otras medidas que sólo miden algunas partes del árbol de dependencias, como la **precision** y la **cobertura** para arcos a la izquierda, a la derecha o a la raíz. Recientemente, algunos investigadores están buscando formas de evaluar los analizadores a través de distintos dominios y haciendo posbile la comparación de tecnologías diferentes (Tsarfaty, Nivre, and Andersson, 2011; Tsarfaty, Nivre, and Andersson, 2012).

[4]Las palabras evaluables pueden variar desde una tarea competitiva a otra. En 2006 (CoNLL-X), no se incluyeron los signos de puntuación, sin embargo, en 2007 si se incluyeron.

### Conclusión

Finalmente, se debe mencionar que en esta tesis, hemos usado principalmente modelos basados en transiciones, básicamente los que están incluidos en MaltParser (Nivre, Hall, and Nilsson, 2006),[5] pero también hemos utilizado MSTParser (McDonald et al., 2005; McDonald, Lerman, and Pereira, 2006) y otros analizadores, siempre con la idea de comparar. En uno de los trabajos de aplicación hemos utilizado Minipar (Lin, 1998), que es un analizador basado en reglas.

## 8.2 Objetivos

Esta tesis intenta, utilizando sistemas del estado del arte (principalmente MaltParser) derivados de las tareas competitivas de las conferencias CoNLL (2006 y 2007), mostrar algunas contribuciones estudiando en primer lugar el análisis de dependencias.

En segundo lugar, algunos estudios y maneras de mejorar la precisión de los analizadores modificando el flujo de análisis y tratando algunos segmentos de las frases de manera separada. Y, más importante, modificando el comportamiento de los analizadores del estado del arte.

En tercer lugar, hemos investigado la selección de features de manera automática y la optimización de los analizadores para los modelos basados en transiciones que consideramos un problema importante y algo que realmente debe realizarse para conseguir resolver de manera más precisa todos los problemas estudiados anteriormente.

En paralelo, y en cuarto lugar, hemos aplicado el análisis de dependencias para resolver algunos problemas interesantes del lenguaje natural.

Por lo tanto, durante el desarrollo de esta tesis, hemos intentado dar respuesta a las siguientes preguntas:

- ¿Qué factores del corpus de entrenamiento pueden afectar la precisión en la fase de análisis?
- ¿Es posible mejorar la precisión manipulando el corpus de entrenamiento?
- ¿Es posible mejorar la precisión combinando diferentes analizadores cuyo origen es el mismo generador de analizadores?
- ¿Es posible modificar el comportamiento de los modelos basados en transiciones con la intención de mejorar su comportamiento?
- ¿Es posible modificar el comportamiento de los modelos basados en grafos con la intención de mejorar su comportamiento?

[5] Mirar, Sección 9.1.1 para encontrar una descripción completa de MaltParser.

- ¿Es posible automatizar la optimización de los analizadores de dependencias?
- ¿Es posible crear un sistema preciso y automático de selección de features para un sistema basado en transiciones? o incluso, ¿un sistema basado en aprendizaje automático?
- ¿Podemos utilizar estructuras de dependencias para resolver problemas del procesamiento del lenguaje natural?

Hemos obtenido resultados interesantes tratando de contestar a esas preguntas durante los últimos años. La presente tesis contesta a las preguntas y en el Capítulo 7, resumimos las respuestas una a una.

## 8.3 Estructura de la Tesis

Los siguientes capítulos contestan a las preguntas presentadas en la Sección 8.2:

- El Capítulo 2 (9 en la versión en castellano) (*Trabajo Relacionado*) muestra el trabajo relacionado de esta tesis, centrándose en métodos estadísticos para el análisis de dependencias. El Apéndice A está relacionado con el Capítulo 2.
- El Capítulo 3 (10 en la versión en castellano) (*Analizando el Análisis de Dependencias*) muestra nuestros primeros estudios sobre analizadores, estudiando principalmente la homogeneidad de su comportamiento y los corpora. El Apéndice B está relacionado con el Capítulo 3.
- El Capítulo 4 (11 en la versión en castellano) (*Mejorando el Análisis de Dependencias*) muestra nuestras ideas para mejorar la precisión del análisis con un sistema de combinación de analizadores, y con más éxito, modificaciones del comportamiento de modelos basados en transiciones y modelos basados en grafos.
- El Capítulo 5 (12 en la versión en castellano) (*Optimizando el Análisis de Dependencias*) muestra nuestro trabajo en optimización del análisis y el entrenamiento, donde explicamos nuestro sistema publicado (MaltOptimizer) que es capaz de encontrar una configuración óptima para un sistema generador de analizadores basados en transiciones (MaltParser). Los Apéndices C y D están relacionados con el Capítulo 5.
- El Capítulo 6 (13 en la versión en castellano) (*Aplicando el Análisis de Dependencias*) muestra algunos trabajos paralelos al resto, donde enfatizamos la gran utilidad de las estructuras de dependencias. El Apéndice E está relacionado con el Capítulo 6.

- El Capítulo 7 (14 en la versión en castellano) (*Conclusiones y Trabajo Futuro*) muestra las conclusiones de la tesis y sugerencias de trabajo futuro.

Todo la investigación incluida en esta tesis está intrinsecamente conectada. Por ejemplo, el problema de la combinación de analizadores[6] y también el trabajo sobre la aplicación de estructuras de dependencias está relacionado con el trabajo sobre la optimización de analizadores. Ya que pueden ser aplicados en sinergia en trabajo futuro utilizando los sistemas automáticos de combinación entrenando nuevos modelos para el resto de problemas, donde la optimización se hizo de manera manual y fue uno de los mayores problemas para llevar a cabo. Una de las razones es que utilizando nuestros métodos automáticos podemos obtener mejores modelos de features en muchos casos, ya que el espacio de búsqueda es demasiado grande para ser explorado de manera manual.

## 8.4 Resumen del Capítulo

En este capítulo hemos introducido los conceptos básicos necesarios para seguir el resto del manuscrito. Además, hemos establecido los objetivos de la tesis y la estructura de la tesis.

[6]Mirar la Sección 11.1, donde mostramos nuestro estudio combinando diferentes analizadores.

# Chapter 9

# Trabajo Relacionado

En este Capítulo mostramos el trabajo relacionado que es relevante para esta tesis.

## 9.1 Análisis de Dependencias

Como se mencionó de manera breve en la introducción, a día de hoy hay varias aproximaciones para la resolución del problema del análisis de dependencias. Los analizadores basados en reglas (o *rule-based*) son una referencia clásica, y están basados en una gramática predefinida, teniendo Minipar (Lin, 1998) como un claro ejemplo. Estos analizadores son muy dependientes del lenguaje para el que fueron desarrollados teniendo que desarrollar una nueva serie de reglas para analizar frases de un lenguaje distinto.

Sin embargo, en los últimos años, los sistemas basados en aprendizaje automático se han convertido en los más populares, como se describe por Buchholz and Marsi (2006) y Nivre et al. (2007). El aprendizaje automático permite tener un analizador para cada lenguaje siempre que tengamos un corpus con el que entrenar. Como se dijo en la Sección 8.1.2, estos analizadores fueron establecidos después de la celebración de las CoNLL Shared Tasks en dos aproximaciones diferentes: **graph-based** (basados en grafos) y **transition-based** (basados en transiciones).

Podemos encontrar sistemas que integran ambas aproximaciones en un único sistema e invitan al usuario a seleccionar entre ellas, como es el caso del analizador presentado por Bernd Bohnet (2010; 2012). Más aún, y como veremos en la Sección 9.3, hay algunos sistemas que integran la salida de ambos sistemas, donde un analizador aprende del otro y viceversa con el objetivo de obtener una mejor precisión (McDonald and Nivre, 2011).

En las siguientes Subsecciones, describimos ambas tecnologías, mostrando un ejemplo de análisis.

### 9.1.1 Análisis de Dependencias Basado en Transiciones

Estos sistemas se llaman *transition-based* (o basados en transiciones), ya que reducen el problema del análisis de una frase a encontrar un camino óptimo a través de un sistema abstracto de transiciones, o máquina de estados. Los sistemas basados en transiciones aprenden modelos que predicen la siguiente acción del estado actual del sistema, incluyendo features (o atributos) que tienen en cuenta el análisis previo y la frase de entrada. El parser comienza con un estado inicial y cambia al resto de estados de manera voraz, utilizando para ello las predicciones del modelo, hasta que se alcanza un estado final. MaltParser pertenece a este tipo de sistemas.

Aquí se muestran las asunciones básicas de esta metodología:

- El analizador procesa la frase de izquierda a derecha de manera determinista.
- Tiene normalmente dos estructuras de datos: una pila y un buffer.
- En cada paso, el algoritmo de análisis debe seleccionar una de las siguientes operaciones: left-arc, right-arc, shift (reduce) o swap. Las operaciones se hacen desde/hacia los nodos que están en la pila hacia/desde los nodos que están en el buffer.
- Un clasificador basado en aprendizaje automático se utiliza para seleccionar la operación en cada paso de análisis.
- Los features (o atributos) basados en categorías gramaticales, morfología, etc, se utilizan para que el sistema de aprendizaje automático seleccione la operación en la mejor manera posible.

En la Figura 9.1 se muestra un ejemplo sencillo de análisis con un analizador basado en transiciones.

Como dijimos arriba, una de las ventajas principales de este tipo de análisis es la eficiencia, con MaltParser es posible realizar análisis en tiempo lineal para estructuras proyectivas y en tiempo cuadrático para estructuras no proyectivas (Bosco et al., 2010). La principal razón es básicamente que el espacio de búsqueda es muy pequeño, básicamente el clasificador debe seleccionar entre las posibles transiciones, normalmente no más de 4.

Merece la pena destacar que para la mayor parte de los estudios mostrados en esta tesis, hemos usado esta perspectiva basada en transiciones.

#### MaltParser

MaltParser (Nivre et al., 2006a; Nivre et al., 2007) es un generador de analizadores de dependencias basado en una máquina de transiciones, donde usando un corpus de entrenamiento anotado con dependencias, es posible

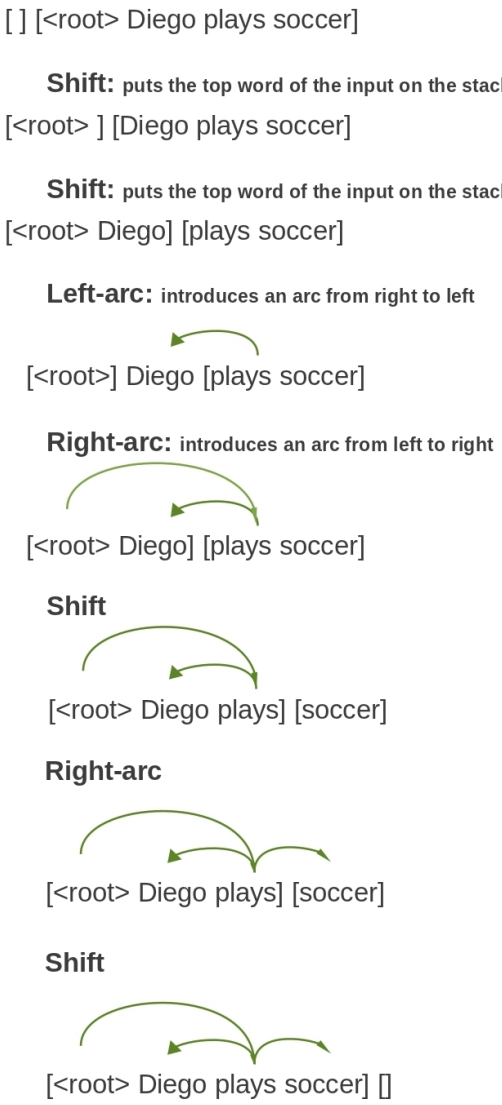



Figure 9.1: Análisis de la frase *Diego plays soccer* con un analizador basado en transiciones. La estructura de datos (entre corchetes) a la izquierda de la imagen es la pila, y la que se encuentra a la derecha es el buffer.

entrenar un modelo capaz de analizar una frases con relaciones de dependencia. Implementa un sistema determinista basado en transiciones para anotar frases con sus árboles de dependencias usando un clasificador que selecciona la mejor transición posible.

Para el entrenamiento, MaltParser utiliza support vector machines (Cortes and Vapnik, 1995), que pueden ser o bien LIBSVM (Chang and Lin, 2001) o LIBLINEAR (Fan et al., 2008), a elección del usuario. Consiste en la

selección de la transición más plausible (crear arco, shift, reduce, etc.) para cada nodo.

En MaltParser existen cuatro familias distintas de algoritmos, todas ellas basadas en transiciones. Proporcionan formas diferentes de generar los arcos, estructuras de datos distintas y formas diferentes de tratar estructuras de dependencias no proyecticas. Las familias son las siguientes:

1. **Algoritmos de Nivre**: incluyen el algoritmo **arc-eager** y el algoritmo **arc-standard**, ambos son versiones del algoritmo descrito por Nivre (2003) y Nivre (2004). La diferencia principal entre ellos, es la manera de generar arcos hacia la izquierda, donde el algoritmo Nivre arc-eager es voraz generándolos tan pronto como puede, a diferencia del algoritmo no voraz arc-standard.

2. **Algoritmos de Covington**: incluyen la versión proyectiva y la versión no proyectiva del algoritmo descrito por Covington (2001) y adaptado por Nivre (2008). Los algoritmos de Covington son similares a Nivre arc-eager, si tenemos en cuenta el orden de análisis.

3. **Algoritmos de Stack (o algoritmos de pila)**: incluyen la versión proyectiva y no proyectiva de los algoritmos descritos por Nivre (2009) y Nivre, Kuhlmann y Hall (Nivre, Kuhlmann, and Hall, 2009). Son similares a Nivre arc-standard, si tenemos en cuenta el orden de análisis.

4. **Algoritmos Multiplanar** (Gómez-Rodríguez and Nivre, 2010): incluyen dos algoritmos; el Planar y el 2-Planar. Son algoritmos capaces de ejecutar en tiempo lineal que cubren el conjunto de estructuras multiplanar descritas por Yli-Jyrä (2003): mientras que el algoritmo Planar se limita a grafos sin arcos que se crucen, que son un superconjunto de los grafos proyectivos, el algoritmo 2-planar permite grafos que pueden ser divididos en 2 planos, permitiendo de ese modo, estructuras no proyectivas.

Las familias de algoritmos de Covington, Stack y Multiplanar contienen algoritmos capaces de analizar estructuras no proyectivas, la familia de algoritmos de Nivre no puede. Sin embargo, cualquier algoritmo proyectivo puede ejecutarse en combinación con un algoritmo que simula estructuras pseudo proyectivas, siendo capaz de generar los arcos proyectivos después del análisis.

MaltParser utiliza modelos de características (o features) que pueden basarse en el árbol de dependencias y la cadena de entrada. Se utilizan para predecir la siguiente acción del análisis cuando el sistema de transiciones alcanza un estado cualquiera.[1]

[1] Ver *http://maltparser.org/userguide.html*

### 9.1.2 Análisis Basado en Grafos

Los sistemas de análisis de dependencias basados en grafos resuelven el mismo problema que los sistemas basados en transiciones pero de manera diferente. La idea principal es que básicamente el analizador genera todos los posibles arcos entre los elementos léxicos, y después un clasificador selecciona los arcos que tienen mayor probabilidad. De ese modo, el analizador extrae de este super grafo, las estructuras de dependencias que conforman los árboles de dependencias de la frase.

La Figura 9.2 muestra un ejemplo simple de un análisis producido por un analizador basado en grafos.

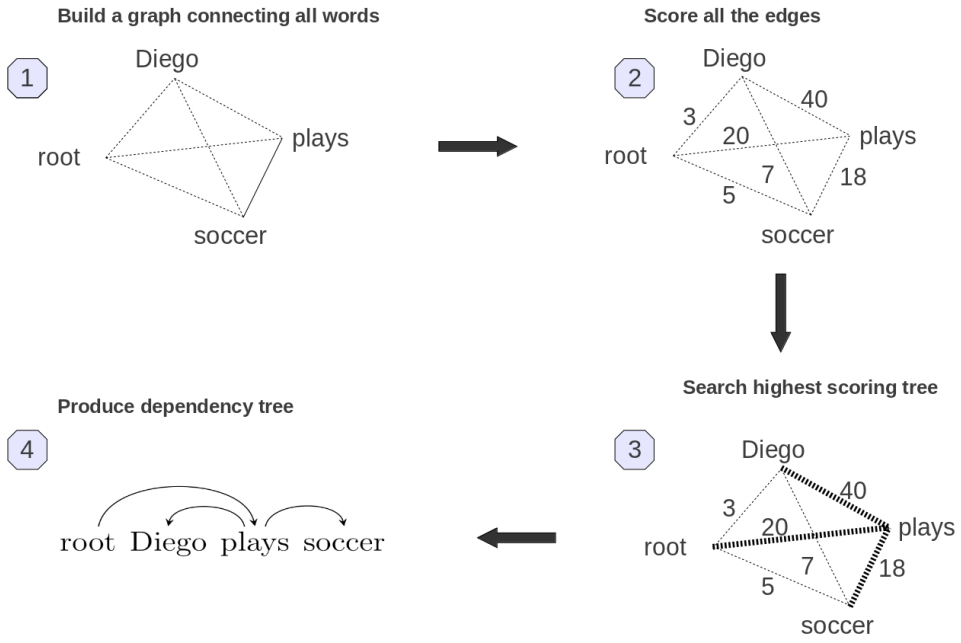



Figure 9.2: Análisis de la frase *Diego plays soccer* con un analizador basado en grafos.

Este tipo de análisis requiere normalmente más tiempo de ejecución que los sistemas basados en transiciones, más memoria e incluso pueden requerir problemas de acceso a disco. Este hecho es el cuello de botella de esta perspectiva. En el ejemplo mostrado en la Figura 9.2 sólo se observan 6 posibles arcos, si sólo tenemos en cuenta los arcos no dirigidos, sin embargo podemos imaginar como de grande es el espacio de búsqueda con frases que tengan más palabras. De hecho, normalmente es cúbico e incluso mayor en tiempo de ejecución.

## 9.2 La CoNLL–X y la CoNLL 2007 Shared Tasks

La CoNLL 2006 (comunmente conocida como CoNLL-X) y la CoNLL 2007 Shared Tasks[2] (Buchholz and Marsi, 2006; Nivre et al., 2007) fueron llevadas a cabo en el contexto de la conferencia de Natural Language Learning (CoNLL). Como se mencionó en la Sección 8.1, sirvieron para generar un formato de entrada común para los analizadores y unas líneas base comunes, que llevó a establecer el estado del arte.

Los dos analizadores que produjeron los mejores resultados en ambas Shared Tasks fueron MSTParser (basado en grafos) y MaltParser (basado en transiciones). Esto es básicamente la razón principal por la que utilizamos ambos analizadores en los experimentos que se llevan a cabo en esta tesis.

Una de las contribuciones principales de estas Shared Tasks es el formato de datos, que se acepta actualmente como un estándar general para todos los analizadores sintácticos de dependencias, e incluso analizadores semánticos en las CoNLL Shared Tasks de 2008 y 2009 (Surdeanu et al., 2008; Hajič et al., 2009).

En la CoNLL-X hubo 13 corpora diferentes y 10 en la CoNLL 2007. Los corpora que comparten lenguaje en ambas tareas son del mismo origen, pero con ligeras modificaciones que mejoran la anotación. Estos corpora son muy importantes para la presente tesis, donde estudiamos todos ellos y los utilizamos en distintos experimentos a lo largo de la tesis. Como se describe por Buchholz and Marsi (2006), Nivre et al. (2007), y Abeillé (2003) los corpora de la CoNLL–X (2006) y la CoNLL 2007 Shared Task son los siguientes:

- **Alemán** (2006). El Tiger Treebank (Brants et al., 2002) contiene frases en alemán de origen periodístico del *Frankfurter Rundschau*.
- **Árabe** (2006 y 2007). El Prague Arabic Dependency Treebank (PADT) (Hajič et al., 2004; Smrž, Šnaidauf, and Zemánek, 2002) que contiene textos en árabe moderno y estándar.
- **Búlgaro** (2006). El Bulgarian dependency treebank (Bultreebank) (Simov, Popova, and Osenova, 2002; Simov et al., 2002; Simov, Osenova, and Slavcheva, 2004; Simov et al., 2005) contiene textos en Búlgaro.
- **Catalán** (2007). El Catalan section of the CESS-ECE Syntactically and Semantically Annotated Corpora (Martí et al., 2007), fue en sus orígenes un árbol de constituyentes, pero fue convertido automáticamenet a estructuras de dependencias por Lluís Màrquez and Antònia Martí.
- **Checo** (2006 and 2007). El Prague Dependency Treebank (PDT) (Böhmová et al., 2003) consiste en un gigantesco volumen de frases en checo, anotadas con dependencias e información morfológica.

[2] Mirar apéndice A donde se muestran los resultados de la CoNLL-X Shared Task

- **Chino** (2006 and 2007). El Sinica Treebank (Chen et al., 2003) es un corpus sintáctico anotado con frases simples en chino.
- **Esloveno** (2006). El Slovene Dependency Treebank (SDT) (Džeroski et al., 2006) contiene frases extraídas de la tradución de *1984* de George Orwell al esloveno.
- **Español** (2006). Se llama AnCora (3LB) (Civit and Antònín, 2002; Navarro et al., 2003; Civit et al., 2003; Palomar et al., 2004) y consiste en textos periodísticos y literarios.
- **Danés** (2006). El Danish Dependency Treebank (Kromann, 2003). La anotación de este corpus de libre acceso, incluye estructuras gramaticales e información sintáctica.
- **Griego** (2007). El Greek Dependency Treebank (Prokopidis et al., 2005) contiene frases en griego moderno.
- **Holandés** (2006). El Dutch Alpino Treebank (Van der Beek et al., 2002b; Van der Beek et al., 2002a) se deriva de un subconjunto del corpus Eindhoven e incluye categorás gramaticales y estructuras de dependencias.
- **Húngaro** (2007). El Szeged treebank (Csendes et al., 2005) contiene textos de ficción y de origen periodístico.
- **Inglés** (2007). Las secciones 2-11 y la seccion 23 del Penn Treebank (Marcus, Marcinkiewicz, and Santorini, 1993) convertidas a dependencias por Ryan McDonald.
- **Italiano** (2007). El Italian Syntactic-Semantic Treebank (ISST) (Montemagni et al., 2003) contiene textos del *Corriere della Sera* y utiliza información de distintos dominios: de lenguaje general y textos financieros.
- **Japonés** (2006). El Japanese Verbmobil Treebank (Kawata and Bartels, 2000) es un corpus anotado sintácticamente basado en diálogos espontáneos.
- **Portugués** (2006). El Bosque part of the Floresta Sintá(c)tica (Afonso et al., 2002) consiste en textos en Portugués (de origen brasileño) automáticamente anotados por el parser *Palavras* (Bick, 2000).
- **Sueco** (2006). El Talbanken05 (Nilsson, Hall, and Nivre, 2005) consiste en prosa profesional, ensayos y entrevistas en sueco.
- **Turco** (2006 y 2007). El Metusabanci Treebank (Oflazer et al., 2003; Atalay, Oflazer, and Say, 2003) consiste en frases en turco que fueron tomadas del METU Turkish Corpus.

- **Vasco** (2007). El Basque dependency treebank (3LB) (Aduriz et al., 2003) contiene textos literarios y de noticias en vasco.

De cada uno de estos corpora se extrajo un pequeño conjunto de test y un conjunto de entrenamiento en las CoNLL Shared Tasks. El tamaño del corpora de entrenamiento es diferente ya que así lo decidieron los autores, pero el tamaño de todos los corpora de test es similar. En la Tabla 9.1 se muestran características sobre estos corpora.

Table 9.1: Número de frases, número de palabras y porcentaje frases no-proyectivas y porcentaje de arcos a la izquierda para cada corpus de las CoNLL Shared Tasks

| **Idioma** | **#Frases (k)** | **#Palabras (k)** | **% frases no proyectivas** | **% arcos a la izquierda** |
|---|---|---|---|---|
| Arabic (2006) | 1.4 | 54.3 | 11.2 | 82.9 |
| Arabic (2007) | 2.9 | 112.0 | 10.1 | 79.2 |
| Basque (2007) | 3.2 | 51.0 | 26.2 | 44.5 |
| Bulgarian (2006) | 12.8 | 190.2 | 5.4 | 62.9 |
| Catalan (2007) | 15.0 | 431.0 | 2.9 | 60.0 |
| Chinese (2006) | 57.0 | 337.0 | 0.0 | 24.8 |
| Chinese (2007) | 57.0 | 337.0 | 0.0 | 24.7 |
| Czech (2006) | 72.7 | 1,249.4 | 23.2 | 50.9 |
| Czech (2007) | 25.4 | 432.0 | 23.2 | 46.9 |
| Danish (2006) | 5.1 | 94.3 | 15.6 | 75.0 |
| Dutch (2006) | 13.3 | 195.0 | 36.4 | 46.5 |
| English (2007) | 18.6 | 447.0 | 6.7 | 49.0 |
| German (2006) | 39.2 | 699.6 | 27.8 | 50.9 |
| Greek (2007) | 2.7 | 65.0 | 20.3 | 44.8 |
| Hungarian (2007) | 6.0 | 132.0 | 26,4 | 27,4 |
| Italian (2007) | 3.1 | 71.0 | 7,4 | 65,0 |
| Japanese (2006) | 17.0 | 151.4 | 5.3 | 8.9 |
| Portuguese (2006) | 9.0 | 206.6 | 18.9 | 60.3 |
| Slovene (2006) | 1.5 | 28.7 | 22.2 | 47.2 |
| Spanish (2006) | 3.3 | 89.3 | 1.7 | 60.8 |
| Swedish (2006) | 11.0 | 191.4 | 9.8 | 52.8 |
| Turkish (2006) | 4.9 | 57.5 | 11.6 | 6.2 |
| Turkish (2007) | 5.6 | 65.0 | 33.3 | 3.8 |

## 9.3 Métodos Híbridos

En el Capítulo 11, concretamente en la Sección 11.1, mostramos un estudio de viabilidad para un analizador que combina distintos analizadores. Debido a este trabajo, creemos necesario mostrar algunos trabajos relacionados y contribuciones interesantes que son importantes para esta tesis.

Los analizadores basados en combinaciones de analizadores pueden dividirse en los siguientes conjuntos:

1. Combinación de analizadores mediante stacking: Nivre y McDonald (2011) usaron stacking para integrar un sistema basado en grafos

(MSTParser) y un sistema basado en transiciones (MaltParser). Stacking es un método donde un analizador aprende de otro, utilizando la salida de ambos parsers y utilizándola para el entrenamiento del otro mediante el uso de una pila. Este trabajo inspiró el nuestro para desarrollar el trabajo mostrado en la Sección 11.1.

2. Combinación de analizadores mediante sistema de votación: Zeman y Zabokrtský (2005) propusieron una aproximación que combina distintos analizadores de dependencias para el checo. La idea es que cada analizador vota cual es la opción más apropiada para cada palabra, y de ese modo, la opción más votada es la seleccionada y se construye el árbol.

3. Combinación de analizadores por descomposición dual: Sagae y Lavie (2006), usaron un sistema basado en grafos y Koo et al. (2010) usó descomposición dual para combinar el algoritmo de Chu–Liu Edmonds[3] con un algoritmo dinámico de tercer orden.

Otro trabajo relevante en la mejora de analizadores de dependencias es (Chen et al., 2009), que muestra una aproximación donde se analizaron frases en inglés y chino generando un sistema que combina analizadores. Estudiaron las preposiciones y como coordinar conjunciones de manera similar a nuestro trabajo presentado en la Sección 11.1.

## 9.4 Optimización de Analizadores y Aprendizaje Automático

En el Capítulo 12, mostramos nuestro sistema que optimiza modelos de parsing de manera automática, es por ello que consideramos relevante incluir algo de trabajo relacionado sobre aprendizaje automático y selección automática de features lo que está directamente relacionado con nuestro trabajo.

La selección automática de features ha sido recientemente explorada por Nilsson y Nugues (2010). Generando un sistema que explora los features de MaltParser empezando desde un modelo de características vacío, y añadiendo features de manera incremental usando la noción de los vecinos topológicos en el espacio de features. Hicieron experimentos con distintos niveles de voracidad y obtuvieron resultados competitivos en tres conjuntos de datos distintos.

En el contexto del procesamiento del lenguaje natural, se han estudiado diversos sistemas de optimización automática como Kool, Zavrel, and Daelemans (2000) y Daelemans et al. (2003), donde se usaron algoritmos

[3] http://www.softpanorama.org/Algorithms/Digraphs/mst.shtml

genéticos para la selección de modelos en el contexto de la anotación de categorías gramaticales y otros sistemas basados en PLN. Estudiaron la selección de features junto con la optimización de parámetros para la optimización de algoritmos de aprendizaje. Existe una herramienta especifica que es capaz de hacer una búsqueda de parámetros en sistemas de aprendizaje automático, ampliamente utilizada en el PLN, conocida como Paramsearch (van den Bosch, 2004).

En aprendizaje automático, la selección automática de features ha sido objeto de numerosos estudios (Guyon and Elisseeff, 2003; McCallum, 2003), y métodos voraces como los que mostramos nosotros en el Capítulo 12 están bien representados en la literatura. Por ejemplo, (Korycinski et al., 2003) usan un sistema voraz adaptativo en el área de la ingeniería óptica, o (Doraisamy et al., 2008) que presenta un estudio comparativo de sistemas de selección de features aplicados a la clasificación automática de géneros. Más aún, (Pahikkala, Airola, and Salakoski, 2010) mostró como acelerar una búsqueda de features hacia delante aplicando una búsqueda voraz, usando una estrategia similar a la que presentamos en el Capítulo 12. Finalmente, (Das and Kempe, 2011) demostraron que los algortimos voraces funcionan bien incluso cuando los features están muy correlacionados y esto es algo que definitivamente pasa en análisis de dependencias.

## 9.5 Resumen del Capítulo

Esta tesis está principalmente basada en sistemas de parsing basados en transiciones, más concretamente, está totalmente relacionada con **MaltParser** (Nivre, Hall, and Nilsson, 2006). Este generador de analizadores está en constante renovación y es de código libre. Pero también hemos investigado otros analizadores, un buen ejemplo es MSTParser que es un parser basado en grafos, o Minipar que es un analizador basado en reglas.

En el campo de la combinación de analizadores hay muchos trabajos que inspiraron nuestro trabajo, como el desarrollado por Joakim Nivre y Ryan McDonald integrando un sistema basado en grafos y un sistema basado en transiciones (2008; 2011), además de comparando y combinando los resultados de ambos sistemas.

En el contexto de optimización de analizadores hay también trabajos relevantes relacionados con el nuestro, como el desarrollado por (Nilsson and Nugues, 2010) para una selección automática de features. Y sistemas más generales como Paramsearch (van den Bosch, 2004).

En este Capítulo hemos mostrado el trabajo relacionado con esta tesis, de modo que puede ser útil para el lector volver atrás y consultar este capítulo en busca de referencias. Es importante tener en cuenta que los siguientes capítulos también muestran trabajo relacionado que sólo es importante en el contexto específico.

# Chapter 10

# Estudios Iniciales sobre Análisis de Dependencias

En este capítulo mostramos algunos análisis técnicos donde estudiamos el análisis de dependencias, MaltParser y sus limitaciones:

1. Estudio sobre la homogeneidad del análisis de dependencias para el español centrándonos en el tamaño de los corpora y la longitud de las frases. En las Secciones 10.1 y 10.2.

2. Estudio sobre las limitaciones de los corpora de la CoNLL-X Shared Task. En la Sección 10.3.

3. Estudio sobre la reevaluación de los analizadores usando medidas de encaje completo (o complete-match), ya que consideramos que deberían ser al menos mencionadas, en trabajos futuros sobre análisis de dependencias. En la Sección 10.4.

La mayoría de los experimentos que se describen en este Capítulo, en particular en las Secciones 10.1, 10.2 y 10.3 fueron llevados a cabo con MaltParser en la configuración por defecto y los modelos de features publicados en la CoNLL-X Shared Task.[1]

En los experimentos donde mostramos la salida de diferentes analizadores, como el que se muestra en la Sección 10.4, solamente usamos las salidas de los analizadores publicadas en la CoNLL-X Shared Task y de ese modo llevar a cabo una comparación justa.[2]

[1] En la web de MaltParser (http://www.maltparser.org/conll.html) es posible encontrar los modelos de features utilizados.

[2] Mirar http://ilk.uvt.nl/conll/online_results.tar.bz2

## 10.1 Estudio de la Precisión para el español: La Asunción de Homogeneidad

Nuestro primer objetivo fue estudiar la precisión considerando factores del corpus. Inspirados por Herrera and Gervás (2008) consideramos que el tamaño del corpus de entrenamiento es un área que se debe considerar cuando se estudia la precisión de los analizadores de dependencias. Después de replicar los resultados obtenidos por el grupo de Nivre en la CoNLL–X Shared Task, tuvimos la siguiente hipótesis: *Es posible mejorar la precisión de los analizadores manipulando el corpus de entrenamiento, o al menos, seremos capaces de extraer interesantes conclusiones de los resultados de los experimentos.*

En esta Sección estudiamos la homogeneidad de la precisión, modificando los corpora de entrenamiento y que la selección del mejor subconjunto para el entrenamiento es algo que debe estudiarse.

### 10.1.1 Primer Experimento: ¿Es homogénea la precisión?

Buscando una forma de estudiar el corpora de entrenamietno, planteamos la siguiente pregunta: *¿Podemos esperar los mismos resultados en términos de precisión para cada texto analizado por un modelo entrenado con MaltParser?* Con la idea de responder a esta pregunta, llevamos a cabo el experimento mostrado a continuación.

#### Configuración del Primer Experimento

Lo primero de todo, dividimos el corpus de Español en 21 subconjuntos de 4.500 palabras cada uno. Estos subconjuntos tienen las siguientes características:

- Un tamaño similar, en palabras, al tamaño del corpus de la CoNLL–X Shared Task (4.500 palabras). Esta división fue hecha para poder tener una comparación realista ya que cada subconjunto es usado en el experimento como corpus de test, al menos una vez.
- Cada subconjunto contiene un número similar de frases al resto y a los corpora de test de la CoNLL-X Shared Task. Esto significa que las frases están distribuidas de manera homogénea en los 21 subconjuntos de acuerdo a su longitud. De ese modo hicimos una selección pseudo aleatoria que proporciona divisiones mucho más homogéneas.

Entrenamos MaltParser con cada subconjunto, con lo que obtuvimos 21 modelos preparados para analizar. Con cada modelo analizamos los otros 20 subconjuntos que no fueron usados para entrenar el modelo en sí. De ese modo obtuvimos $21 \times 20 = 420$ corpora analizados. Después de ello,

extrajimos los resultados de cada uno de ellos. Como medidas de evaluación utilizamos no sólo LAS, UAS y LA, si no también las siguientes:

- Los coeficientes de correlación entre:
  - Los resultados de LAS y los resultados de UAS para cada modelo ($r_{LAS,UAS}$).
  - Los resultados de LAS y los resultados de LA para cada modelo ($r_{LAS,LA}$).
  - Los resultados de UAS y los resultados de LA para cada modelo ($r_{UAS,LA}$).

  Obteniendo estos coeficientes, quisimos investigar si, a pesar de las diferencias entre los subconjuntos analizados, existía una cierta correlación entre cada par de métricas.

- Los valores máximos y mínimos para cada caso:
  - LAS ($max_{LAS}$, $min_{LAS}$).
  - UAS ($max_{UAS}$, $min_{UAS}$).
  - LA ($max_{LA}$, $min_{LA}$).

Quisimos, además, estudiar la estabilidad de la precisión conseguida por MaltParser cuando se entrena con un corpus de la CoNLL-X Shared Task, en este caso, el español.

### Resultados del Primer Experimento

La Tabla 10.1 muestra los resultados de evaluación de los 20 corpora evaluados considerando coeficientes de correlación, y los valores máximos y mínimos.

Si analizamos la segunda columna de la Tabla ($r_{LAS,UAS}$) podemos concluir que en la mayoría de los casos, la correlación entre LAS y UAS es elevada. Podemos obtener la misma conclusión con LAS y LA mirando los resultados de la tercera columna de la tabla ($r_{LAS,LA}$). Estos valores, son lógicos si consideramos las definiciones de LAS, UAS y LA. Los resultados para LAS dependen simultáneamente de que el arco de dependencias esté correctamente conectado y además un etiquetado correcto. El arco correcto es lo que afecta a UAS y el etiquetado afecta a LA. Pero cuando analizamos la cuarta columna ($r_{UAS,LA}$) observamos diferentes valores, desde 0.44 a 0.91; puede también explicarse considerando las definiciones de UAS y LA. UAS no depende del etiquetado. A pesar de que estos números parecen lógicos, si el analizador es suficientemente bueno, debería ser capaz de generar bien tanto el arco como la etiqueta.

| **Sub–corpus** | $\mathbf{r}_{LAS,UAS}$ | $\mathbf{r}_{LAS,LA}$ | $\mathbf{r}_{UAS,LA}$ | $\mathbf{max}_{LAS}$ | $\mathbf{min}_{LAS}$ | $\mathbf{max}_{UAS}$ | $\mathbf{min}_{UAS}$ | $\mathbf{max}_{LA}$ | $\mathbf{min}_{LA}$ |
|---|---|---|---|---|---|---|---|---|---|
| $A_0$ | 0.84 | 0.78 | 0.44 | 72.78% | 69.06% | 78.22% | 74.20% | 85.03% | 82.45% |
| $A_1$ | 0.87 | 0.85 | 0.65 | 72.86% | 68.81% | 77.29% | 74.88% | 84.85% | 82.36% |
| $A_2$ | 0.92 | 0.87 | 0.91 | 73.55% | 68.40% | 78.47% | 69.04% | 85.85% | 75.24% |
| $A_3$ | 0.92 | 0.80 | 0.54 | 72.68% | 69.01% | 77.58% | 74.42% | 85.39% | 82.35% |
| $A_4$ | 0.88 | 0.87 | 0.71 | 72.02% | 68.96% | 77.09% | 74.31% | 84.71% | 82.45% |
| $A_5$ | 0.82 | 0.89 | 0.61 | 72.74% | 69.32% | 77.32% | 74.90% | 84.99% | 82.40% |
| $A_6$ | 0.90 | 0.88 | 0.69 | 71.90% | 68.42% | 76.88% | 74.32% | 84.78% | 82.20% |
| $A_7$ | 0.86 | 0.88 | 0.63 | 72.55% | 68.16% | 77.27% | 73.61% | 85.24% | 81.95% |
| $A_8$ | 0.94 | 0.87 | 0.71 | 72.92% | 67.71% | 77.55% | 73.65% | 85.52% | 82.03% |
| $A_9$ | 0.91 | 0.76 | 0.56 | 72.27% | 68.23% | 77.47% | 73.99% | 84.85% | 82.35% |
| $A_{10}$ | 0.87 | 0.89 | 0.69 | 71.76% | 68.19% | 77.11% | 73.00% | 84.62% | 81.74% |
| $A_{11}$ | 0.91 | 0.85 | 0.69 | 73.30% | 68.37% | 78.27% | 73.68% | 85.73% | 82.52% |
| $A_{12}$ | 0.92 | 0.78 | 0.60 | 73.01% | 69.27% | 78.32% | 74.62% | 85.39% | 82.43% |
| $A_{13}$ | 0.89 | 0.85 | 0.64 | 72.96% | 69.61% | 78.22% | 74.46% | 85.60% | 82.19% |
| $A_{14}$ | 0.92 | 0.81 | 0.62 | 73.04% | 68.29% | 77.93% | 74.07% | 85.04% | 82.27% |
| $A_{15}$ | 0.94 | 0.85 | 0.76 | 71.37% | 67.81% | 76.21% | 72.60% | 85.01% | 82.42% |
| $A_{16}$ | 0.87 | 0.76 | 0.44 | 72.51% | 68.83% | 76.83% | 73.89% | 85.50% | 82.17% |
| $A_{17}$ | 0.92 | 0.79 | 0.58 | 73.23% | 68.82% | 77.58% | 74.17% | 85.78% | 82.77% |
| $A_{18}$ | 0.95 | 0.78 | 0.63 | 72.50% | 67.40% | 77.63% | 73.18% | 84.70% | 81.82% |
| $A_{19}$ | 0.82 | 0.86 | 0.54 | 72.25% | 68.99% | 77.65% | 74.19% | 85.39% | 82.99% |
| $A_{20}$ | 0.95 | 0.84 | 0.73 | 72.73% | 68.28% | 77.55% | 73.68% | 85.19% | 82.07% |
| **max** | 0,95 | 0,89 | 0,91 | **73,55%** | **69,61%** | **78,47%** | **74,90%** | **85,85%** | **82,99%** |
| **avg** | 0,90 | 0,83 | 0,64 | **72,62%** | **68,57%** | **77,54%** | **73,76%** | **85,20%** | **81,96%** |
| **min** | 0,82 | 0,76 | 0,44 | **71,37%** | **67,40%** | **76,21%** | **69,04%** | **84,62%** | **75,24%** |

Table 10.1: Resultados obtenidos por los modelos entrenados con los 21 subconjuntos en los que se dividió el corpus de español.

Si consideramos los 21 modelos, el que muestra la máxima variación de los valores de LAS es el $A_8$ con una diferencia de 5.21 puntos. El que muestra la mínima variación es $A_4$ con una diferencia de 3.06 puntos. Para UAS, la máxima variación la produce $A_2$ con una diferencia de 9.43 puntos, y $A_1$ produce la mínima variación obteniendo una diferencia de 2.41 puntos. Finalmente, $A_2$ alcanza la máxima diferencia no sólo para UAS sino también para LA (10.61 puntos), mientras que la mínima diferencia ocurre de nuevo con el modelo $A_4$ con 2.26 points, que además obtuvo la mínima variación para LAS. Teniendo en cuenta estos valores podemos concluir que cada modelo puede alcanzar un rango amplio de precisión dependiendo de los textos que se usan como entrada. Pero los resultados de los 21 modelos, son más homogéneos, como se puede ver en las últimas tres filas de la tabla, desde la quinta columna hasta la última.

### Conclusiones del Primer Experimento

Después de analizar los resultados presentados en la subsección anterior, podemos concluir que la precisión es suficiéntemente homogénea. Por lo tanto:

- Cuando entrenamos dos modelos con corpora diferentes que tienen un tamaño similar y un número similar de frases para cada longitud,

deben alcanzar valores promedio similares. Sin embargo, cada modelo podría tener diferencias notables en la precisión dependiendo del conjunto de test. Por ello, pensamos que el tamaño de los corpora y la longitud de las frases pueden contribuir a la precisión. Y ello nos llevó a generar los experimentos descritos en las Secciones 10.1.2 y 10.1.3.

- Mejores valores de precisión pueden ser alcanzados combinando analizadores específicos. Cada analizador específico analizaría sólo subconjuntos para los que produce buenos resultados. Esta idea motiva el experimento de la Sección 10.1.2 y el analizador de N-Versiones mostrado en la Sección 11.1.

### 10.1.2 Segundo Experimento: ¿Es Posible Conseguir un Mejor Resultado?

Los resultados obtenidos cuando analizamos un subconjunto con diferentes modelos nos inspiró para realizar otro estudio utilizando las mismas divisiones. Este segundo experimento fue configurado para verificar si algunos modelos inducen ruido en la precisión y para encontrar respuesta a la siguiente pregunta: *¿Hay modelos que producen mejores resultados que otros?* Si la respuesta a la pregunta es que sí, podemos concluir que los corpora de entrenamiento deben ser anotados con mucho cuidado para alcanzar la mejor precisión posible dada la tecnología disponible.

#### Configuración del Segundo Experimento

Para este segundo experimento usamos las metricas computadas por los 420 modelos del experimento anterior. Si todos los $A_i$ se analizan con un modelo entrenado con $A_j, j \neq i$, que produce el mejor resultado posible (para LAS), entonces podremos obtener la mejor media de LAS para el corpus concreto, $\bigcup_{0 \leq i \leq 20} A_i$. En pocas palabras, el mejor promedio para LAS puede ser obtenido combinando la acción de todos los modelos, de modo que todos los modelos analizarán sólo los subconjuntos para los que son los mejores. La Tabla 10.2 muestra que modelo debe usarse para analizar cada $A_i$.

#### Resultados del Segundo Experimento

Como se muestra en la Tabla 10.2 algunos modelos, entrenados con conjuntos específicos, son capaces de analizar mejor que otros modelos con subconjuntos diferentes. Por ejemplo, el modelo entrenado con $A_{17}$ es el mejor analizando 4 de los 21 subconjuntos. Pero hay otros modelos que siempre devuelven peores resultados que otros. Podría ser porque esos modelos contienen estructuras sintácticas muy diversas y requieren mayor similitud.

| Input | Parsed by model | LAS |
|---|---|---|
| $A_0$ | $A_{14}$ | 72.58 % |
| $A_1$ | $A_{18}$ | 72.20 % |
| $A_2$ | $A_{17}$ | 70.93 % |
| $A_3$ | $A_1$ | 70.99 % |
| $A_4$ | $A_{12}$ | 72.47 % |
| $A_5$ | $A_{13}$ | 70.17 % |
| $A_6$ | $A_2$ | 73.55 % |
| $A_7$ | $A_0$ | 71.58 % |
| $A_8$ | $A_4$ | 71.57 % |
| $A_9$ | $A_{17}$ | 73.23 % |
| $A_{10}$ | $A_3$ | 71.91 % |
| $A_{11}$ | $A_3$ | 71.78 % |
| $A_{12}$ | $A_3$ | 71.27 % |
| $A_{13}$ | $A_7$ | 69.83 % |
| $A_{14}$ | $A_6$ | 71.90 % |
| $A_{15}$ | $A_5$ | 70.24 % |
| $A_{16}$ | $A_5$ | 70.07 % |
| $A_{17}$ | $A_1$ | 70.77 % |
| $A_{18}$ | $A_{12}$ | 71.94 % |
| $A_{19}$ | $A_{17}$ | 71.40 % |
| $A_{20}$ | $A_{17}$ | 70.79 % |
| **Avg LAS** | | 71.48 % |

Table 10.2: Modelos entrenados con $A_j, j \neq i$ que deben ser usados para analizar cada $A_i$ para obtener el mejor LAS.

#### Conclusiones del Experimento

Los resultados discutidos aquí pueden motivar una evaluación más completa de los analizadores de depedendencias, que no sólo mediría el LAS del analizador si no la persistencia de la precisión a través de un conjunto de textos distintos.

Como vemos que algunos modelos son mejores que otros y analizan mejor algunos subconjuntos, este experimento debe motivar el desarrollo de analizadores n–versiones; mirar la Sección 11.1. Estos analizadores consistirían en diversos modelos específicos, cada uno entrenado para obtener un buen resultado para un tipo concreto de frases. Por lo tanto, el sistema debería seleccionar el modelo que mejor va a analizar la frase en concreto que en ese momento se va a tratar.

Además, como conclusión podemos sugerir que en el desarrollo de corpora se puede evitar esfuerzo innecesario incluyendo frases que ya están incluidas en otras, y seleccionando aquellas que son más convenientes.

### 10.1.3 Tercer Experimento: El Tamaño del Corpus de Entrenamiento Afecta a la Precisión

Esta subsección presenta un experimento centrándose en el análisis del efecto en la precisión del tamaño del corpus de entrenamiento.

### Configuración del Tercer Experimento

Para analizar el efecto del tamaño del corpus de entrenamiento en la precisión construímos incrementalmente un corpus y evaluamos para cada modelo entrenado, como sigue:

- Lo primero de todos, seleccionamos el $A_i$ $(1 \leq i \leq 20)$ para el que obtuvimos el mejor LAS cuando analizamos todos los $A_i$ $(1 \leq i \leq 20)$ con el modelo entrenado con $A_0$ (ver el experimento descrito en la Sección 10.1.2). Este subconjunto fue $A_6$ y fue el primero que se añadió al corpus incremental, que inicialmente estaba vacío.

- En cada iteración entrenamos MaltParser con el corpus incremental y lo analizamos con el subconjunto $A_0$.

- En cada iteración añadimos el siguiente $A_i$ $(1 \leq i \leq 20)$ para el que obtuvimos mejor LAS cuando lo entrenamos con $A_0$ en el experimento descrito en la Sección 10.1.2.

- Iteramos 20 veces hasta que cada $A_i$ fue añadido al corpus incremental. Cada $A_i$ $(1 \leq i \leq 20)$ tiene una distribución de frases proporcional teniendo en cuenta la longitud de las frases, de ese modo en cada iteración el corpus incremental tenía una longitud media de frase similar.

### Resultados del Tercer Experimento

Los resultados del tercer experimento se muestran en la Figura 10.1. Desde la primera a la segunda iteración el valor de LAS mejora en casi 3 puntos. Desde la segunda a la tercera iteración mejora en 1.38 puntos. En la cuarta iteración mejora en 1.2 puntos. Y mejora casi 1 punto después de la quinta y la sexta iteración. Añadiendo 22.600 palabras al corpus de entrenamiento que teníamos en la primera iteración obtuvimos un incremento por tanto de 7.56 puntos. Pero añadiendo otras 22,600, el valor de LAS sólo se incrementó en 1.63 puntos. Teniendo en cuenta las variaciones de LAS mostradas en la Sección 10.1.2, este último incremento de 22.600 palabras no es muy significativo. Con lo que podría significar que después de un cierto límite el tamaño del corpus de entrenamiento no contribuye a la precisión del análisis.

### Conclusiones del Tercer Experimento

Después de estudiar de manera sistemática los incrementos de precisión obtenidos cuando se amplia el tamaño del corpus de entrenamiento mientras se mantiene la distribución de la longitud de las frases, podemos concluir que después de un cierto límite la cantidad de palabras no afecta de manera significativa la precisión. En otras palabras, un corpus de entrenamiento

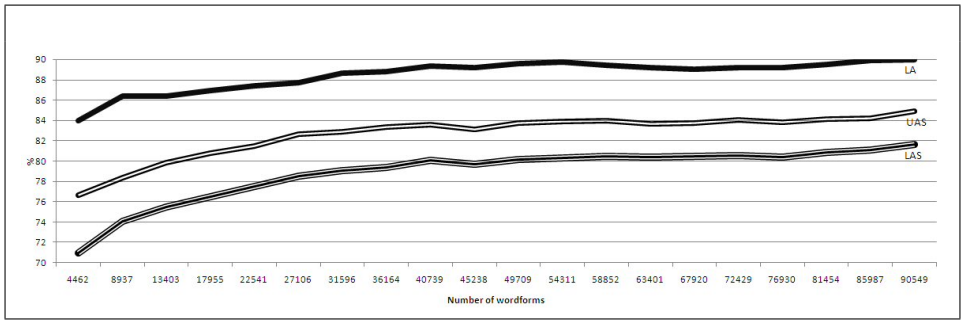



Figure 10.1: LAS, UAS y LA dependiendo del número de palabras contenidas en el corpus de entrenamiento.

que contiene todos los tipos de frases posibles no contribuye a la precisión después de un cierto límite. Esto significa que, al menos para el español, las palabras no son tan importantes como las frases en los corpora de entrenamiento. Este hecho significa que cuando se construye corpora de entrenamiento se debe mirar el tipo de frases a incluir en vez de buscar un tamaño importante al final. Estas ideas nos llevaron a estudiar el corpora de entrenamiento de manera más profunda en las Secciones 10.2 y 10.3.

## 10.2 El Rol de la Longitud de las Frases

Como se demostró por McDonald y Nivre (2011), los sistemas de análisis tienden a perder precisión en las frases más largas, debido a sus estructuras sintácticas complejas. En esta Sección queremos mostrar que las frases más largas son más útiles si consideramos incluirlas en los corpora de entrenamiento con la idea de conseguir una mejor precisión.

Después del experimento previo (donde vemos que la longitud de las frases se revela como una característica importante en los corpora de entrenamiento para el español) y considerando el trabajo hecho por McDonald y Nivre (2011) donde los errores de los analizadores relacionados con la longitud de las frases, decididimos llevar a cabo otro experimento centrándonos en la longitud de las frases. Por lo tanto, la idea es determinar si la longitud de las frases contenidas en el corpus de entrenamiento pueden afectar la precisión del análisis. Realmente, este experimento consiste en 2 experimentos distintos que se describen a continuación.

El primero, descrito en la Subsección 10.2.1, muestra un estudio sobre el efecto de la longitud de las frases en la precisión. El segundo, descrito en la Subsección 10.2.2 compara la precisión de los corpora construidos exclusivamente con frases largas y los corpora construidos exclusivamente con frases cortas.

### 10.2.1 Corpora de Entrenamiento Conteniendo Frases de Longitud Única

Este experimento muestra un estudio sobre el efecto de la longitud de las frases, basándose en la siguiente hipótesis: *las frases más largas son más útiles que las cortas para el entrenamiento.*

La Figura 10.2 muestra la distribución de las frases en el corpus de español de acuerdo a la longitud de las frases; en el eje x representamos la longitud de las frases y en el eje y el número de frases. Merece la pena destacar que las frases largas son muy raras en el corpus. Con lo cual, nos surge otra pregunta: *¿podemos mejorar la precisión construyendo un corpus con frases de longitud única?*

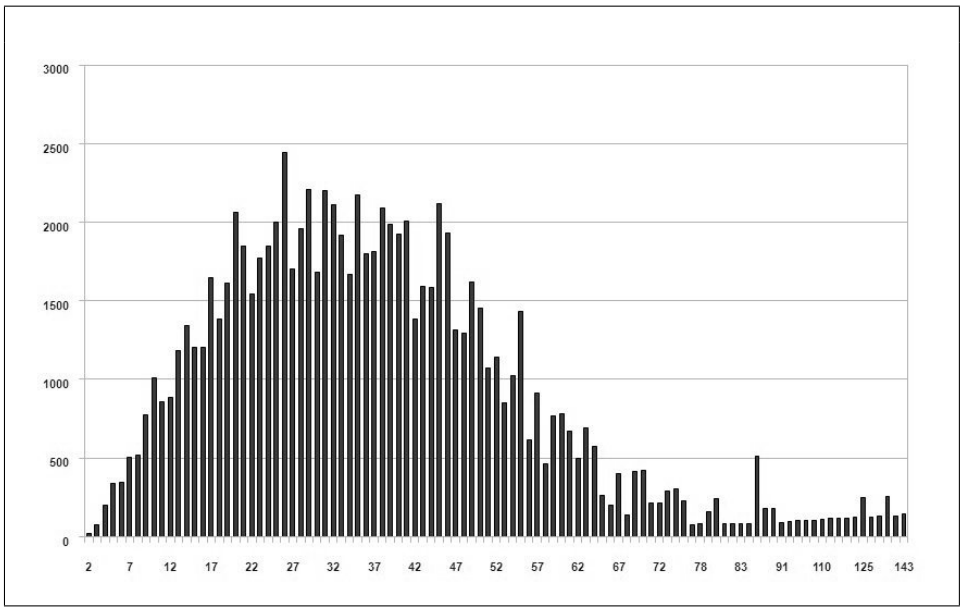



Figure 10.2: Distribución de las frases en el corpus de español de acuerdo a su longitud.

#### Configuración del Experimento

La sección del corpus de español que se utilizó para el entrenamiento en la CoNLL–X Shared Task la dividimos en 102 subconjuntos, cada uno conteniendo frases de longitud única. Con lo que al final teníamos, un subconjunto de 1 frase de 143 palabras, otro subconjunto de 1 frase con 130 palabras, otro subconjunto de 2 frases de 128 palabras y así sucesivamente.

Entrenando MaltParser con cada uno de estos subconjuntos, obtuvimos 102 modelos distintos. Con cada uno, analizamos el conjunto de test de la CoNLL-X Shared Task.

### Resultados del Experimento

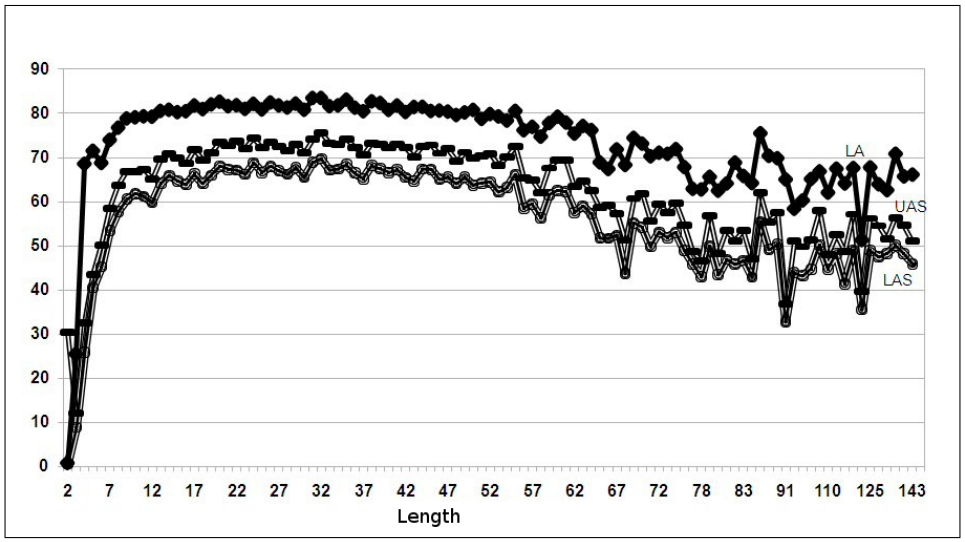



Figure 10.3: LAS, UAS y LA cuando entrenamos con corpora que contiene frases de longitud única.

Los resultados del experimento se muestran en la Figura 10.3. En esta figura, mostramos LAS, UAS y LA para cada análisis. En el eje x, representamos la longitud de las frases contenidas en el subconjunto usado como corpus de entrenamiento. Los valores de LA se muestran en la línea con mejores resultados (la de arriba), los valores de UAS se representan por la línea de en medio y los valores de LAS por la de abajo.

Como puede observarse, los corpora de entrenamiento que contienen frases más largas producen una precisión importante a pesar de su tamaño limitado. Por ejemplo, el corpus de 143 palabras, que contiene una única frase, produce 45.88% LAS, 51.16 % UAS y 66.15% LA evaluándolo con todo el corpus de test.

Teniendo en cuenta los resultados mostrados arriba y el hecho de que el corpus de español tiene 35 frases con 80 ó más de 80 palabras[3], llevamos a cabo un nuevo experimento tratando de dar respuesta a la siguiente pregunta: *¿Podríamos obtener mejores resultados considerando sólo las frases largas?* Por lo tanto, lo que hicimos fue ir añadiendo de manera sistemática frases, creando nuevos subconjuntos de entrenamiento, y entrenamos un modelo con cada subconjunto. Empezamos con las frases de longitud 120 ó superior, después repetimos el mismo experimento con frases de 110 palabras que añadimos al corpus previo. Después las frases de más de 100 palabras; después las de más de 90 palabras y finalmente las de más de 80 palabras. La Tabla 10.3 muestra LAS, UAS y LA para estos subconjuntos.

[3]La figura 10.2 muestra que las frases más largas son una pequeña parte del corpus.

| Subcorpus | Size | LAS | UAS | LA |
|---|---|---|---|---|
| 120..143 | 1154 | 61.07% | 67.60% | 77.18% |
| 110..143 | 1612 | 64.64% | 70.23% | 80.24% |
| 100..143 | 1919 | 66.68% | 71.99% | 81.41% |
| 90..143 | 2104 | 66.86% | 72.17% | 81.90% |
| 80..143 | 3538 | 68.60% | 74.11% | 82.72% |

Table 10.3: Resultados obtenidos por modelos entrenados con los subconjuntos, sólo las frases más largas (tamaño en palabras).

| Subcorpus | Size | LAS | UAS | LA |
|---|---|---|---|---|
| 2..7 | 1471 | 57.60% | 62.19% | 78.10% |
| 2..8 | 1991 | 62.16% | 66.86% | 79.75% |
| 2..9 | 2765 | 64.51% | 68.90% | 81.52% |
| 2..10 | 3775 | 66.84% | 72.30% | 82.79% |

Table 10.4: Resultados obtenidos por los modelos entrenados con los subconjuntos, sólo las frases más cortas (tamaño en palabras).

Los resultados son notablemente elevados, comparables a los obtenidos con corpus más grandes de más de 4,500 palabras que contienen frases de todas las longitudes como vimos en la Sección 10.1.1.

Repetimos el mismo experimento sólo con frases cortas para poder tener una comparativa. Obtuvimos los resultados mostrados en la Tabla 10.4. Hay 571 frases en el corpus de español con 10 palabras ó menos. Como podemos observar en los resultados mostrados en la Tabla 10.3 y la Tabla 10.4 podemos ver como los corpora que contienen las frases más largas son más precisos tanto en la generación de arcos como en el etiquetado de los mismos, debemos mirar el tamaño final de cada subconjunto de entrenamiento para llevar a cabo una comparación justa. Merece la pena tener en cuenta que en cada caso, las diferencias de LAS y UAS son muy remarcables.

### Conclusiones del Experimento

Por los resultados mostrados arriba, puede concluirse que las frases largas cuando se incluyen en el corpus de entrenamiento, contribuyen más que las frases más cortas a la precisión global. Sustancialmente en el caso de la generación de arcos pero con una perdida no muy importante en el etiquetado. Con lo que un corpus de entrenamiento que contenga sólo frases largas necesita menos palabras que un corpus de entrenamiento que contiene sólo frases cortas para alcanzar precisiones similares. Creemos que este hecho es una conclusión muy importante y podría llevar a la construcción de corpora

de manera más eficiente centrándose en la calidad de las frases y la anotación y no en el tamaño final del corpus.

### 10.2.2 Corpora de Entrenamiento que Contiene las Mejores Frases

Después, desarrollamos otro experimento centrándonos también en la longitud de las frases. Nuestro objetivo, fue mostrar que las frases más largas incluidas en los corpora de entrenamiento son útiles para entrenar modelos más precisos que las frases más cortas. El experimento es similar al llevado a cabo en la Sección 10.1.3.

#### Configuración del Experimento

Construimos incrementalmente un corpus de entrenamiento y evaluamos la precisión del análisis para cada modelo entrenado, como sigue:

- Lo primero de todo, seleccionamos el subconjunto para el que obtuvimos el mejor LAS en el experimeto de la Sección 10.2.1.
- En cada iteración entrenamos MaltParser con el corpus incremental (como en 10.1.3) y lo evaluamos con el modelo entrenado evaluando la sección de test del corpus usado en la CoNLL-X Shared Task.
- En cada iteración añadimos al corpus incremental la parte no usada del subscorpus para el cual obtuvimos el mejor LAS en el experimento descrito en esta Sección.
- Iteramos 102 veces cubriendo todos los subconjuntos.

#### Resultados del Experimento

En la Figura 10.4 mostramos LAS, UAS y LA para cada análisis. En el eje x representamos el número de palabras contenidas en el corpus incremental. En la Figura observamos como con la mitad del corpus de entrenamiento alcanzamos los valores máximos de precisión, y alcanzamos esos valores máximos mucho más rápido que en el experimento de la Sección 10.1.3, que se resumen en la Figura 10.1.

#### Conclusiones del Experimento

Por los resultados del experimento mostrados arriba, podemos concluir que la selección de los corpora de entrenamiento teniendo en cuenta la longitud de las frases permite alcanzar mejores resultados de LAS, UAS y LA que cuando no se tiene en cuenta. Podemos observar que la precisión se

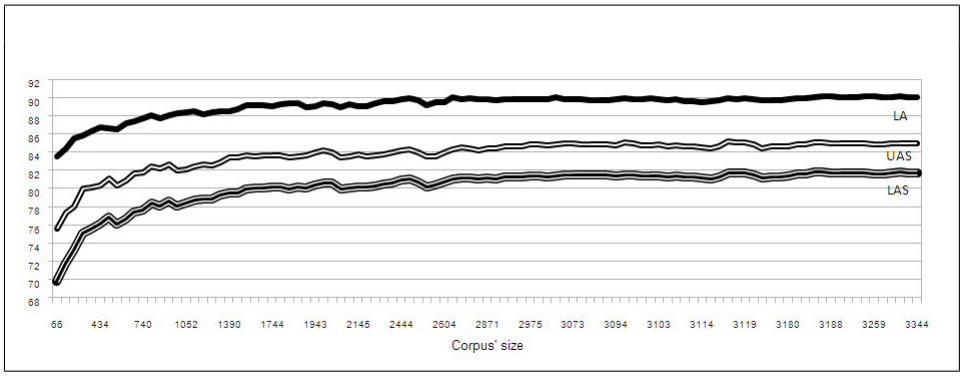



Figure 10.4: LAS, UAS y LA cuando consideramos la longitud de las frases para contruir un corpus de entrenamiento.

estabiliza con la mitad de las frases del corpus. Esto significa que los modelos generados con los corpora conteniendo las frases más cortas no son tan buenos para el entrenamiento como los modelos generados con las frases más largas. Si comparamos los resultados de este experimento, mostrado en la Figura 10.4 y los resultados del experimento 10.1.3, mostrados en la Figura 10.1, la precisión crece mucho más rápido en este segundo experimento que en el primero. La razón es que en este segundo experimento, con una configuración similar, seleccionamos en primer lugar las frases más largas.

## 10.3 Estudio del Corpora de Entrenamiento para Múltiples Lenguajes

Esta sección muestra otro experimento que trata el problema del tamaño de los corpora de entrenamiento para un conjunto importante de lenguajes (los que estuvieron presentes en la CoNLL-X Shared Task). Creemos que el tamaño de los copora siempre es un cuello de botella para el entrenamiento, considerando tiempo y restricciones de memoria. Por lo tanto, este experimento fue motivado por los experimentos de las secciones anteriores, donde estudiamos el mismo problema desde una perspectiva distinta y un único corpus. En el experimento previo nos dimos cuenta que parece que los corpora actuales usados para entrenar modelos basados en aprendizaje automático contienen una proporción significativa de frases que no es necesaria para obtener una buena precisión. En esta Sección demostramos que el corpus de español no es el único caso.

El desarrollo de los corpora de entrenamiento requiere un gran esfuerzo, por ello, considereamos que un proceso apropiado para la selección de las frases que van a ser incluidas puede resultar en la obtención de modelos entrenados que pueden ser tan precisos como los que se entrenan con corpora

más grandes que no han sido previamente optimizados.

En esta Sección queremos intentar demostrar que los corpora de entrenamiento pueden contentener más información que la necesaria para entrenar analizadores de dependencias basados en aprendizaje automático.

### 10.3.1 Nuestra Hipótesis: ¿Por qué Consideramos el Tamaño de los Corpora?

Como mostramos en las Secciones 10.1 y 10.2, los corpora con tamaño similar (con un tamaño parecido en palabras y una distribución parecida de frases de acuerdo a su longitud) usados para entrenar MaltParser para el español, producían modelos que proporcionaban valores similares de precisión tanto mínimos como máximos. Por lo tanto, MaltParser muestra un comportamiento estable. Pero es importante destacar que estos valores máximos y mínimos no son siempre dados para los mismos subconjuntos del texto cuando se analizan para estos modelos. Cada porción de texto que se analiza normalmente se analizaba mejor por algunos modelos (normalmente uno o dos), mientras que el resto de modelos eran mejores para otros textos.

Algunos autores, como Nivre et al. (2007) o Herrera and Gervás (2008) han presentado evidencias de que el tamaño de los corpora de entrenamiento no garantiza una precisión importante por si mismo. Un corpus grande permite estadísticamente la presencia de un número más grande de ejemplos, pero también permite la presencia de elementos que podrían producir ruido cuando los ejemplos de entrenamiento no se seleccionan uno a uno. Además, como vimos en las Secciones 10.1 y 10.2, es probado que la inclusión de nuevas palabras a partir de un cierto umbral no contribuye en la misma medida a mejorar la precisión cuando se entrena MaltParser para el español. Con lo cual, nuestro planteamiento es que los corpora de la CoNLL-X Shared Task podrían tener información que no contribuye del mismo modo análisis cuando se entrena con MaltParser. Teniendo en cuenta que la frecuencia del tipo de estructuras del conjunto de entrenamiento es importante queremos analizar si podríamos eliminar de los corpora algunos ejemplos sin afectar mucho la precisión.

Una reducción de los corpora de entrenamiento produce una importante reducción en el tiempo de ejecución. Teniendo en cuenta que MaltParser es un generador de analizador eficiente, como podemos ver en la Sección 9.1.1, es posible analizar en tiempo lineal para árboles proyectivos y cuadrático para árboles no proyectivos (Bosco et al., 2010), lo que significa que una reducción de un N% de los nodos genera una reducción del N% en el tiempo de ejecución en el caso lineal. Si consideramos el caso cuadrático la reducción del N% de los nodos es más notable. Cuando consideramos miles (o incluso millones) de palabras, esto es absolutamente significativo.

Otro factor importante cuando se construyen estos corpora de entrenamiento es el coste de producción. Como ejemplo del esfuerzo necesario

para construir estos corpora anotados con dependencias, Prokopidis et al. (2005) publicaron el proceso llevado a cabo para el desarrollo del corpus de Griego (Greek Dependency Treebank (GDT)), el cual fue utilizado en la CoNLL Shared Task de 2007. El desarrollo del corpus llevó un mes completo a treinta anotadores diferentes para alcanzar un tamaño final de 70.000 palabras. Podemos suponer que anotar las 3.521.286 palabras (lo que es la suma de todos los corpora de la CoNLL-X Shared Task) fue un proceso muy complejo. Con lo cual, si podemos probar que los corpora reducidos son una manera válida de entrenar analizadores de dependencias, el desarrollo de futuros corpora podría ser optimizado generando mejores corpora de menor tamaño.

En resumen, teniendo en cuenta todo lo anterior, nuestra hipótesis es que *es posible crear corpora efectivo reduciendo todas las frases que contienen información que podría ser redundante y que podría encontrarse en otros ejemplos, cuando esta redundancia no es útil para el sistema entrenado.*

En la siguiente Sección 10.3.2 mostramos el desarrollo de un experimento que demuestra que en los corpora existentes existe una proporción de frases que no aporta mucho al resultado final con respecto a la precisión de los analizadores. Además sugerimos algunas ideas de como podemos reducir el tamaño de los mismos.

### 10.3.2 Demonstrando Nuestra Hipótesis

Para analizar el efecto del tamaño del corpus de entrenamiento en la precisión construimos un corpus incremental (para cada lenguaje) y evaluamos la precisión en el análisis para cada modelo entrenado. De ese modo, el experimento es similar al que se llevó a cabo en las Secciones 10.1 y 10.2,

#### Diseño del Experimento

Dividimos cada corpus en 15 subconjuntos pequeños. Los primeros 5 utilizando el 50% de las palabras de cada corpus y el resto usando el 50% restante de cada corpus:

Esto significa que para cada corpus dividimos:

- El primer conjunto formado por el primer 50% de las palabras en 5 subconjuntos, cada uno conteniendo el 10% de las palabras.
- El segundo conjunto formado con el segundo 50% de las palabras fue dividido en 10 subconjuntos pequeños conteniendo un 5% de las palabras cada uno.

Cada subconjunto fue seleccionado respetando la longitud media por frase de todo el corpus, con lo que llevamos a cabo una selección pseudo aleatoria de las frases. Esto significa que todos los subconjuntos contienen,

más o menos, el mismo número de palabras y la longitud media de las frases es, en practica, la misma para cada subconjunto.

Usando estos 15 subconjuntos, nuestro experimentos se llevó a cabo como sigue:

- En cada iteración añadimos el siguiente subconjunto que no utilizado empezando con los corpora que contienen el 10% de las palabras y terminando con los que tienen el 5% construyendo el corpus inicial en la última iteración (100%).

- Entrenamos con el corpus incremental y evaluamos con la seccón de los corpora que se utilizó como conjunto de test en la CoNLL–X Shared Task.

- Iteramos mientras existen subconjuntos que permanecen sin utilizar.

### Resultados del Experimento

En esta Sección mostramos los resultados del experimento descrito arriba, mostrando el resultado en LAS en cada iteración.

Los resultados se muestran en la Tabla 10.5, y en la Figura 10.5 mostramos el comportamiento mediante un gráfico.

| **Language** | 10% | 20% | 30% | 40% | 50% | 55% | 60% | 65% | 70% | 75% | 80% | 85% | 90% | 95% | 100% |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Arabic | 1.69 | 51.08 | 59.40 | 61.63 | 62.11 | 63.14 | **63.01** | 63.66 | 64.66 | 65.02 | 66.23 | 66.60 | **66.33** | 67.57 | **67.42** |
| Bulgarian | 10.81 | 75.70 | 82.97 | 84.90 | 85.17 | 85.97 | 85.89 | 86.21 | 86.59 | **86.59** | 86.94 | 86.96 | 87.40 | **87.38** | 87.57 |
| Chinese | 62.09 | 77.57 | 81.12 | 82.82 | 84.06 | 84.77 | **84.71** | **84.55** | 85.17 | 85.59 | 85.97 | 86.63 | **86.38** | 86.69 | 86.99 |
| Czech | 0.88 | 71.00 | 72.98 | 74.96 | **74.96** | 75.44 | 75.70 | 76.12 | **75.8** | 76.26 | **76.18** | 76.42 | 76.92 | **76.92** | 77.04 |
| Danish | 11.80 | 77.07 | 80.18 | 81.30 | 81.40 | 82.33 | 82.76 | **82.66** | 83.19 | 83.59 | 83.80 | 83.90 | 84.29 | **84.02** | 84.51 |
| Dutch | 9.10 | 66.05 | 68.40 | 71.57 | 72.19 | 73.35 | 73.90 | **72.97** | **73.79** | **73.63** | **73.43** | 74.73 | **74.73** | 75.01 | **74.47** |
| German | 8.74 | 80.50 | 82.29 | 82.74 | 82.60 | 83.46 | 83.81 | 84.03 | 84.22 | **83.42** | 84.70 | **84.70** | 84.98 | 85.31 | 85.33 |
| Japanese | 18.43 | 87.77 | 89.17 | 89.80 | 90.37 | 90.47 | 90.63 | 91.05 | 91.07 | 91.21 | **91.16** | **91.16** | **90.96** | 91.52 | 91.88 |
| Portuguese | 12.29 | 75.75 | 77.17 | 78.99 | 79.08 | 79.59 | **79.57** | 79.91 | 80.23 | **80.13** | 80.78 | 81.12 | **81.10** | 81.21 | **81.21** |
| Slovene | 4.92 | 49.72 | 54.20 | 58.81 | 59.75 | 61.36 | 62.77 | 63.06 | 64.04 | **63.91** | **64.00** | 64.07 | **63.98** | 64.88 | 65.56 |
| Spanish | 46.36 | 72.25 | 75.98 | 79.30 | **78.96** | 79.78 | 79.86 | **79.64** | 80.22 | 80.75 | **80.36** | **80.75** | 81.55 | **81.39** | 81.87 |
| Swedish | 10.21 | 73.67 | 75.85 | 76.62 | 78.13 | 78.94 | 80.14 | 81.25 | 82.06 | 82.10 | 82.56 | 83.28 | **83.12** | **83.17** | 83.50 |
| Turkish | 12.45 | 58.77 | 61.47 | 61.98 | 62.83 | 63.16 | 63.25 | 63.42 | 63.50 | 63.68 | 63.76 | 64.32 | 64.58 | 64.69 | 64.84 |
| Average | 17.48 | 76.41 | 80.1 | 82.12 | 82.63 | 83.48 | 83.83 | 84.04 | 84.55 | 84.66 | 84.99 | 85.39 | 85.53 | 85.81 | 86.02 |

Table 10.5: Resultados generales (LAS) obtenidos por los modelos iterativos con los corpora de tamaño reducido. Mostramos en negrita los casos en los que el resultado es peor (o el mismo) que una iteración previa.

Podemos concluir que después de un cierto límite la cantidad de palabras que se añaden a los corpora no afecta a la precisión en la misma medida. En otras palabras, un corpus de entrenamiento que contiene un rango variado de longitudes de frase, no contribuye del mismo modo a la precisión después de un cierto límite. Parece más importante incluir todas las estructuras sintácticas posibles. Estos resultados pueden llevar a la comunidad a desarrollar corpora de manera más eficiente

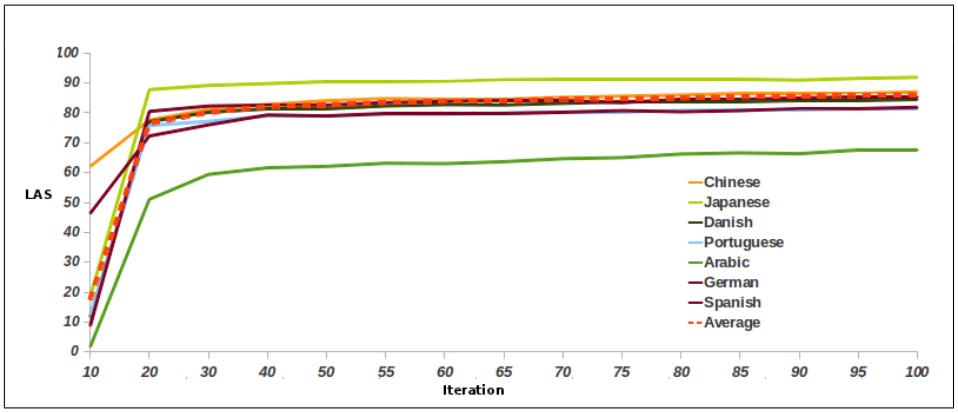



Figure 10.5: Comportamiento estable que todos los corpora muestran considerando LAS.

### Análisis de los Resultados

Como se ve en la Figura 10.5 y la Tabla 10.5, una vez que se alcanza un tamaño significativo (alrededor del 50%), las mejoras conseguidas al añadir más palabras no son muy grandes. Como se ve en la Tabla 9.1 en la Sección 9.2, el 50% de las frases varía entre 624.704 en el caso mayor (Checo) y 14.375 en el más pequeño (Esloveno).

Considerando el caso extremo del corpus Checo, PDT (Czech Prague Dependency Treebank), entrenar un modelo con las primeras 624.704 palabras (50%), la precisión del modelo entrenado alcanza el 74,96% LAS, mientras que con el corpus completo , alcanza 77,04% LAS. Por lo tanto, se consigue una mejora de 74,96 puntos percentuales con las primeras 624.704 palabras y utilizando las segundas 624.704 es sólo de 2.08 puntos. Lo mismo pasa para todos los idiomas, teniendo en cuenta que otros lenguajes no tienen corpora tan grandes como en el caso del Checo.

Todos los datos mostrados en negrita en la Tabla 10.5, muestran los casos en los que un modelo entrenado con un corpora más pequeño obtiene mejores resultados (o los mismos) que uno obtenido con un corpora de mayor tamaño.

## 10.3.3 Conclusiones

Debido a los resultados del experimento, donde se muestra que los corpora que consisten en sólo el 50% del tamaño original, producen analizadores que están en el mismo rango de precisión que los que se obtienen entrenando con los corpora completos. Por lo tanto, estos resultados deberían animar a desarrollar corpora teniéndolos en cuenta. Recomendamos seleccionar con cuidado las frases que se van a incluir, de acuerdo a las estructuras sintácticas. Con lo que este estudio deberá ser entendido como una justifi-

cación para el desarrollo de corpora de manera más eficiente, y en definitiva, pensando con cuidado que frases se deben incluir en los nuevos corpora.

Creemos que el futuro desarrollo de corpora debería tratar de evitar lo siguiente:

- Dos frases de la misma longitud que comparten la estructura sintáctica.
- Dos frases, una más corta que la otra, donde la estructura sintáctica de la frase corta está completamente incluida en la frase larga.

Con lo cual, podríamos sugerir las siguientes ideas:

- No repetir estructuras sintácticas, pero teniendo en cuenta que podría ser también útil no repetir subestructuras.
- Incluir frases en el mayor rango posible de longitudes.

Finalmente, consideramos que la inclusión de estas ideas (o similares) puede llevar a construcción de corpora con menor esfuerzo económico y/o a la extensión de los existentes o su adaptación a nuevos dominios.

## 10.4 Estudio Sobre Medidas de Evaluación Basadas en Frases

Como vimos en la Sección 8.1.2, las medidas de evaluación comunes son LAS, UAS y LA. Estas medidas se utilizaron en las CoNLL Shared Tasks sobre Análisis de Dependencias, y están basadas en medir la precisión token a token. Dado que ambos congresos CoNLL fueron totalmente relevantes en el área, ahora estas medidas de evaluación se han convertido en un estándar *de facto* cuando se evalúan analizadores de dependencias. A pesar de ello, algunos autores como Yamada y Matsumoto (2003) propusieron otro tipo de medidas basadas en computar la precisión frase a frase, ellos las describieron como medidas *Complete Rate*. Más aún, algunos trabajos recientes han usado estas medidas que pueden denominarse como *complete match* para evaluar sus sistemas, como Goldberg y Elhadad (2010).

Por lo tanto, esta Sección tiene el objetivo de atraer atención a estas medidas, ya que de ese modo se obtiene una descripción más rica de la precisión de los analizadores, cuando se necesita, combinándolas con medidas basadas en tokens. Con este objetivo, en esta Sección mostramos una reevaluación de los 19 parsers que participaron en la CoNLL-X Shared Task evaluándolos con los 13 corpora de evaluación que se utilizaron.

### 10.4.1 Medidas Sentence-Based

Consideramos, los textos analizados como conjuntos de frases. Es por ello, que computamos medidas de evaluación que pueden tener en cuenta tanto el

grafo de dependencias sin etiquetar o el grafo con etiquetas de dependencias para cada frase en el conjunto de test. Además de ello, consideramos medidas macro, que permiten otro punto de vista sobre medidas basadas en token añadiendo información por frase. En definitiva, en este estudio utilizamos las siguientes medidas:

- Macro–Average LAS (MacroLas) es el porcentaje de tokens que se miden en el conjunto de test con un correcto etiquetado y una correcta conexión con su nodo padre, generando una media frase a frase.
- Labeled Complete–Match (LCM) es el porcentaje de frases con un grafo (etiquetado) perfectamente generado.

### 10.4.2 Reevaluando los Parsers de la CoNLL-X Shared Task con Medidas Sentence-Based

| Parser | Arab | Bulg | Chin | Czech | Dan | Dutch | Germ | Japa | Port | Slov | Span | Swed | Turk | Tot |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **McD.** | **71.11** | 88.29 | 88.40 | **82.24** | 85.95 | 80.35 | **89.13** | 95.43 | 87.63 | **75.96** | **83.58** | 85.33 | 75.06 | **83.73** |
| Niv. | 70.33 | 88.61 | 89.56 | 79.87 | **86.38** | **80.96** | 88.08 | **96.06** | **88.45** | 71.02 | 82.94 | **86.64** | **76.25** | 83.47 |
| O'N. | 71.06 | **86.63** | 89.50 | 78.74 | 83.95 | 79.16 | 87.87 | 95.42 | 85.69 | 73.91 | 81.87 | 84.50 | 70.23 | 82.19 |
| Che.$^{\uparrow}$ | 69.89 | 87.47 | 87.51 | 78.14 | 83.55 | 74.59 | 86.70 | 95.07 | 85.74 | 73.90 | 81.47 | 83.74 | 73.74 | 81.65 |
| Rie.$^{\downarrow}$ | 70.80 | – | **92.13** | 70.77 | 85.26 | 79.39 | 88.62 | 95.40 | 85.37 | 74.25 | 79.17 | 83.26 | 71.03 | 81.29 |
| Sag.$^{\downarrow}$ | 67.47 | – | 87.60 | 78.83 | 83.99 | 77.73 | 87.19 | 95.28 | 87.06 | 72.84 | 78.40 | 84.45 | 74.60 | 81.29 |
| Cor. | 68.33 | 84.48 | 83.05 | 77.08 | 82.54 | 73.90 | 85.33 | 95.12 | 85.63 | 75.14 | 82.40 | 82.37 | 73.13 | 80.65 |
| Car.$^{\uparrow}$ | 65.72 | 84.23 | 86.76 | 71.62 | 81.07 | 70.34 | 84.33 | 94.18 | 84.13 | 71.04 | 79.20 | 81.40 | 70.36 | 78.80 |
| Cha.$^{\downarrow}$ | 58.56 | – | 87.49 | 69.00 | 81.13 | 75.38 | 86.53 | 94.73 | 82.19 | 71.31 | 80.62 | 84.37 | 71.97 | 78.61 |
| Wu.$^{\uparrow}$ | 67.34 | 81.40 | 78.47 | 54.82 | 79.59 | 73.16 | 79.95 | 95.25 | 82.31 | 70.05 | 73.50 | 75.72 | 67.77 | 75.33 |
| Bic.$^{\uparrow}$ | 58.58 | 80.36 | 80.56 | 66.07 | 76.79 | 72.32 | 76.63 | 92.11 | 76.20 | 66.49 | 73.36 | 77.44 | 66.30 | 74.09 |
| Can. | 53.57 | 79.93 | 83.93 | 56.20 | 79.93 | 77.40 | 81.73 | 93.64 | 74.08 | 57.43 | 68.67 | 81.62 | 65.36 | 73.35 |
| Shi.$^{\uparrow}$ | 67.30 | – | – | – | 76.94 | – | – | – | – | 66.33 | 74.84 | 82.10 | 67.13 | 72.44 |
| Joh.$^{\downarrow}$ | 68.68 | – | 74.29 | 71.50 | 81.87 | 74.59 | 81.17 | 87.26 | 84.01 | 68.15 | 76.39 | 78.51 | 72.82 | 70.71 |
| Liu.$^{\uparrow}$ | 56.66 | 69.00 | 80.00 | 61.31 | 80.34 | 63.91 | 72.60 | 84.68 | 72.28 | 60.12 | 66.49 | 67.96 | 53.17 | 68.34 |
| Yur.$^{\downarrow}$ | 49.36 | 75.04 | 78.09 | 47.31 | 73.50 | 69.30 | 67.85 | 92.17 | 66.49 | 53.27 | 71.01 | 68.96 | 71.85 | 68.02 |
| Sch. | 43.14 | – | 71.66 | 51.47 | 76.87 | 72.44 | 72.26 | 91.59 | 66.55 | 49.00 | 48.34 | 74.52 | 61.98 | 64.99 |
| Dre.$^{\downarrow}$ | 53.95 | 74.56 | 76.36 | 62.91 | 66.78 | 66.36 | 73.34 | 91.09 | 74.63 | 61.53 | 66.88 | 68.76 | 56.47 | 63.83 |
| Att.$^{\downarrow}$ | 50.12 | 70.06 | 51.60 | 55.25 | 64.90 | 49.37 | 66.45 | 44.07 | 72.20 | 56.33 | 65.48 | 63.84 | 44.65 | 58.02 |
| Av | 62.21 | 80.77 | 81.50 | 67.40 | 79.54 | 72.81 | 80.88 | **90.48** | 80.04 | 66.74 | 74.45 | 78.71 | 67.57 | *74.78* |

Table 10.6: Resultados de la CoNLL–X Shared Task para Macro–Average LAS (MacroLAS). Las flechas indican si hay una reclasificación entre parsers comparando con los resultados por LAS.

Para ilustrar nuestra propuesta reevaluamos la participación de todos los sistemas de la CoNLL-X[4] computando medidas basadas en frases. Los resultados para MacroLAS y LCM se encuentran en las Tablas 10.6 y 10.7.

Los resultados para LCM rondan el 30%, pero hay que tener en cuenta la dificultad de la tarea, ya que en este caso se mide si se obtiene el grafo completo (y además etiquetado).

Los resultados de MSTParser (el parser de McDonald et al.) y MaltParser (el parser de Nivre et al.) fueron los mejores y muy parejos en

[4]Usando las salidas que se encuentran en la web de la CoNLL-X Shared Task.

| Parser | Arab | Bulg | Chin | Czech | Dan | Dutch | Germ | Japa | Port | Slov | Span | Swed | Turk | Tot |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Niv**$^{\uparrow}$ | 9.59 | **32.91** | 68.05 | 27.12 | **26.09** | **27.46** | 34.73 | **75.32** | **31.60** | 18.41 | **17.96** | **32.13** | 19.26 | **32.36** |
| McD.$^{\downarrow}$ | 9.59 | 30.15 | 62.51 | **27.95** | 24.22 | 25.91 | 34.73 | 72.92 | 23.96 | 18.91 | 17.48 | 27.76 | **19.42** | 30.42 |
| Sag.$^{\uparrow}$ | 8.22 | – | 61.25 | 23.01 | 23.91 | 23.06 | **36.69** | 71.23 | 27.78 | **20.40** | 12.62 | 27.76 | 19.10 | 29.59 |
| Che.$^{\uparrow}$ | 9.59 | 29.15 | 59.63 | 23.01 | 20.19 | 19.43 | 32.49 | 71.51 | 20.83 | 18.91 | 14.08 | 26.48 | 17.50 | 27.91 |
| Rie.$^{\downarrow}$ | 9.59 | – | **72.09** | 13.42 | 21.12 | 22.28 | 32.49 | 71.65 | 21.53 | 13.93 | 10.19 | 23.91 | 13.80 | 27.17 |
| O'N.$^{\downarrow}$ | 9.59 | 26.63 | 62.63 | 20.55 | 18.94 | 21.50 | 31.93 | 71.79 | 21.18 | 15.17 | 11.17 | 25.96 | 13.32 | 26.95 |
| Cor. | **10.27** | 23.87 | 46.83 | 20.82 | 16.15 | 18.65 | 28.85 | 71.79 | 22.57 | 18.16 | 15.05 | 24.16 | 15.25 | 25.57 |
| Cha. | 2.74 | – | 61.59 | 1.64 | 17.39 | 19.95 | 34.17 | 71.51 | 19.10 | 5.72 | 15.05 | 27.25 | 14.44 | 24.21 |
| Car.$^{\uparrow}$ | 8.22 | 20.10 | 58.71 | 17.26 | 15.22 | 17.36 | 25.21 | 67.70 | 19.79 | 14.68 | 15.53 | 20.82 | 13.80 | 24.18 |
| Wu.$^{\uparrow}$ | 8.22 | 23.62 | 47.29 | 0.00 | 13.35 | 17.88 | 24.37 | 72.21 | 21.18 | 13.43 | 7.77 | 14.91 | 11.71 | 21.23 |
| Bic.$^{\uparrow}$ | 8.22 | 13.82 | 43.83 | 11.51 | 10.87 | 18.13 | 17.37 | 62.20 | 4.51 | 7.71 | 9.22 | 16.97 | 10.11 | 18.04 |
| Can. | 0.00 | 14.07 | 46.25 | 0.00 | 12.11 | 18.91 | 22.97 | 65.73 | 0.00 | 0.00 | 4.85 | 18.25 | 10.11 | 16.40 |
| Joh.$^{\downarrow}$ | 8.22 | – | 33.10 | 7.94 | 11.94 | 15.80 | 16.25 | 50.63 | 14.58 | 7.96 | 6.31 | 14.40 | 9.47 | 16.38 |
| Liu.$^{\uparrow}$ | 7.53 | 9.30 | 42.10 | 9.59 | 12.11 | 13.99 | 14.29 | 53.74 | 7.99 | 5.22 | 4.85 | 13.11 | 5.62 | 15.34 |
| Yur. | 0.00 | 10.55 | 44.87 | 0.00 | 9.32 | 16.58 | 12.32 | 63.47 | 0.00 | 0.00 | 5.34 | 11.31 | 14.44 | 14.48 |
| Dre.$^{\downarrow}$ | 0.00 | 5.28 | 39.10 | 8.77 | 0.62 | 14.51 | 12.89 | 59.80 | 6.25 | 5.47 | 2.42 | 7.71 | 4.17 | 12.84 |
| Shi.$^{\uparrow}$ | 8.90 | – | – | – | 12.11 | – | – | – | – | 8.21 | 6.31 | 23.91 | 9.15 | 11.43 |
| Sch.$^{\uparrow}$ | 0.00 | – | 40.72 | 0.00 | 3.73 | 13.73 | 8.12 | 0.28 | 0.00 | 0.00 | 0.00 | 9.51 | 0.00 | 6.34 |
| Att.$^{\downarrow}$ | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Av | 6.24 | 18.42 | 49.48 | 11.81 | 14.18 | 18.06 | 23.33 | **59.64** | 14.60 | 10.12 | 9.27 | 19.28 | 11.61 | *20.04* |

Table 10.7: Resultados de la CoNLL–X Shared Task para Labeled Complete Match (LCM). Las flechas indican si hay una reclasificación entre parsers comparando con los resultados por LAS.

la Shared Task. MSTParser es el mejor cuando consideramos MacroLAS debido probablemente a su capacidad de predecir correctamente los arcos, sin embargo MaltParser es el mejor cuando se considera LCM probablemente debido a su capacidad prediciendo las etiquetas de dependencias (McDonald and Nivre, 2011). De nuevo, MaltParser y MSTParser son los mejroes parsers con estas medidas de evaluación y los resultados obtenidos con MacroLAS demuestran que son realmente precisos cuando se mide la precisión frase a frase.

Se observa que los resultados de MaltParser y MSTParser son similares para LCM y MacroLAS y siguen un comportamiento similar para todos los corpora. Por lo tanto, se puede concluir que ambos analizadores son eligibles bajo esta perspectiva.

Estas medidas evidencias que las frases largas son un problema complejo ya que la mayoría de los analizadores muestran dificultades cuando los analizan (McDonald and Nivre, 2011), lo que significa que los lenguajes con una mayor longitud en sus frases están directamente afectados por este hecho. La mayoría de los conjuntos de test contienen frases con mucha variedad en su longitud, con la excepción de Chino y Japonés, donde la media es muy pequeña y la mayoría de las frases son similares en cuanto a su longitud. Este hecho se evidencia muy bien con la medida LCM. Por ejemplo, la longitud media por frase en el corpus de Chino es 5,78 palabras y el resultado de LCM es 49,58. Para el corpus de Árabe, tenemos justo el caso opuesto, 36,80 es el valor medio de sus frases y obtenemos tan sólo 6,24 LCM. En la Figura 10.6 se muestra esta correlación para todos los idiomas.

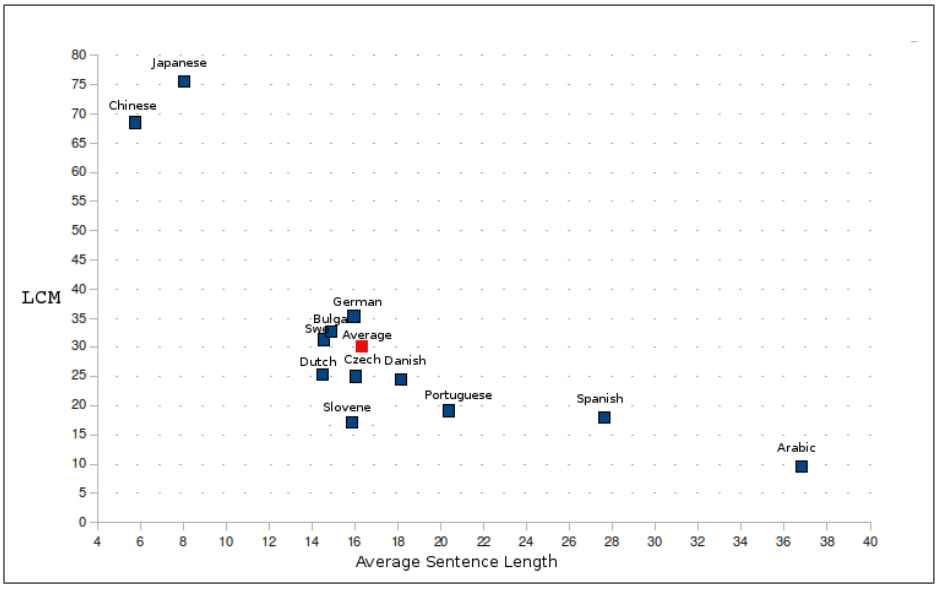



Figure 10.6: Correlación entre la Longitud Media por Frase (*Average Sentence Length*) en los conjuntos de test y los resultados usando la medida LCM cuando se analizan con MaltParser.

### 10.4.3 Conclusiones del Estudio

Teniendo en cuenta los resultados discutidos en la Subsección 10.4.2, el uso de las medidas sentence–based podría dar otra perspectiva a la pregunta de que analizador es mejor para una tarea correspondiente. Si sólo consideramos medidas basadas en los tokens (LAS, UAS y LA) podemos estar perdiendo información que puede ser relevante para tomar esa decisión.

En resumen, es claro que estas medidas deberían utilizarse cuando esperamos una precisión elevada a nivel de frase, y esto normalmente se requiere cuando para tareas donde se analizan árboles de dependencias. Este estudio muestra la importancia de estas medidas de evaluación, y nos gustaría animar a los investigadores a utilizarlas para estudiar la precisión de manera más profunda.

Merece la pena recalcar que la reclasificación de los analizadores es mayor para LCM que para MacroLAS, como se ve en las Tablas 10.6 y 10.7, ya que algunos analizadores tienen dificultades analizando frases largas, por ejemplo el analizador de Attardi et al. no es capaz de analizar por completo ninguna de las frases.

Finalmente, creemos que dados los resultados mostrados en este estudio y donde se observa una fuerte correlación entre LCM y la longitud media por frase, consideramos que se debe generar corpora que permita unos resultados directamente comparables, por ejemplo, teniendo una longitud media por frase similar.

## 10.5 Resumen del Capítulo

Hemos mostrado una serie de experimentos y estudios que nos proporcionaron las ideas iniciales y la motivación suficiente para mejorar, optimizar o aplicar el análisis de dependencias de manera más compleja. Sin estos experimentos iniciales los trabajos explicados en los siguientes Capítulos no podrían haberse llevado a cabo.

En las Secciones 10.1 y 10.2, vimos como se comportan los analizadores y mostramos como tratamos de modificar las posibles salidas, principalmente aprendimos que la calidad de los corpora de entrenamiento es muy importante para el entrenamiento y la precisión final. También concluímos que las frases largas son muy ricas en estructuras sintácticas diversas y esto proporciona mucha información para el entrenamiento y las hace muy útiles.

Del experimento mostrado en la Sección 10.3, aprendimos que los corpora de entrenamiento pueden contener información que podría ser suprimida obteniendo resultados muy parejos a los obtenidos con los corpora completos.

Del último estudio, mostrado en la Sección 10.4, vimos que proporcionando una perspectiva distinta a la evaluación de los analizadores puede aportar información muy relevante sobre el comportamiento de los analizadores.

# Chapter 11

# Mejora del Análisis de Dependencias

Uno de los objetivos de la presente tesis es tratar de mejorar la precisión de los analizadores mostrando algunas contribuciones e ideas originales en ese sentido. En este Capítulo se muestran dos trabajos diferentes con esta intención, mejorando la precisión de analizadores basados en MaltParser.

- El primero (Sección 11.1) es un estudio de viabilidad para un sistema híbrido en el que se combinan analizadores, donde solamente basamos nuestros estudios en el corpus español sin modificar internamente los analizadores. La idea consiste en analizar algunos segmentos de las frases de manera separada, más concretamente, entrenar analizadores específicos para tratar palabras que están normalmente mal analizadas por los modelos entrenados con el corpus completo.

- El segundo (Sección 11.2), está basado en un estudio en profundidad sobre la posición de la raíz durante el tiempo de análisis y el tiempo de entrenamiento. La posición de la raíz se situa normalmente a la izquierda de las frases y la literatura no estudia el problema en profundidad. Sin embargo, en este trabajo, demostramos que este problema es relevante, ya que la situación de la raíz puede producir una mejora en los analizadores cuando se sitúa al final (o a la derecha) de las frases, o incluso no teniendo ninguna raíz.

Por lo tanto, este capítulo esta dividido en dos partes diferenciadas donde se muestran aproximaciones diferentes buscando maneras de mejorar la precisión de los modelos generados por MaltParser.

## 11.1 Estudio de Viabilidad para un Sistema que Combina Analizadores: Hacia un Analizador de N-Versiones

Este estudio se motiva con los resultados del Capítulo 10, donde mostramos evidencias de que el tamaño de los analizadores no está necesariamente asociado con el tamaño del corpus de entrenamiento y la perspectiva alternativa de evaluar los parsers con medidas de precisión por frase (complete-match accuracy).

Basándonos en los resultados actuales obtenidos por MaltParser, el objetivo de este trabajo es mejorar no solamente los resultados globales, pero conseguir una mejor precisión frase a frase. Al final, el usuario final obtendría una solución más satisfactoria. Para ello, tratamos de buscar respuesta a las siguientes cuestiones:

1. ¿Es viable mejorar la precisión combinando analizadores entrenados con corpora muy específicos?

2. ¿Qué acciones pueden automatizarse para mejorar la precisión?

Estas ideas están además inspiradas por McDonald y Nivre (2007; 2011) y diversos sistemas híbridos, como los que se muestran en la Sección 9.3. En esta Sección proponemos un analizador que consiste en varios analizadores, cada uno de ellos capaz de analizar una parte de las frases para después combinar la salida de los analizadores.

Es bueno tener en cuenta, que este estudio, sigue siendo un estudio de viabilidad ya que cuando tratamos de obtener resultados competitivos en el caso real, no fue posible. Sin embargo, creemos interesante mostrar los resultados del experimento ya que proporciona interesantes conclusiones.

En la siguientes subsecciones, discutimos nuestra propuesta para combinar distintos analizadores (N analizadores), todos ellos integrados formando un analizador completo. En la Sección 11.1.1 motivamos el presente estudio de viabilidad, la Sección 11.1.2 muestra un conjunto de experimentos llevados a cabo para confirmar la hipótesis, la Sección 11.1.3 muestra el algoritmo llevado a cabo hacia un analizador de N–versiones.

### 11.1.1 Motivación

Con el corpus en español, a pesar de una elevada precisión global sólo se obtiene un 18.4% LCM, con lo cual si conseguimos aumentar la precisión por frase conseguiremos aumentar la precisión global, y el usuario final obtendrá un resultado más satisfactorio.

Además, hay un conjunto de palabras que normalmente están mal analizadas tanto en el etiquetado como en el arco de dependencias. Estas

palabras son las preposiciones “a”, “de”, “ en”, “con”, “por”, la conjunción (que en español tiene dos palabras distintas: “y” ó “e”), y el nexo“que”. Por ejemplo, hay sólo 20 frases en el corpus con sólo un error en el análisis, y en 10 de esos 20 casos, el fallo lo produce una de estas palabras.

Un análisis incorrecto de estas palabras puede ser la razón de que el árbol dejase de ser útil, ya que estas palabras (preposiciones, conjunciones y nexos), suelen ser la raíz de los subárboles. Con lo que nuestra hipótesis consiste en mejorar el análisis de estas palabras obteniendo un resultado más satisfactorio.

### 11.1.2 Obteniendo N Analizadores de Dependencias

El primer estudio que realizamos fue un analizador de N–versiones simple. Nuestra idea era determinar si algunas palabras ‘difíciles’ de analizar podrían analizarse mejor con analizadores específicos mientras un analizador general analiza el resto. Por lo tanto, nuestro analizador propuesto funciona de la siguiente manera: el analizador general analiza la frase por completo. Si se detecta un patrón que encaja con alguno de los analizadores específicos, el correspondiente analizador específco asociado a ese patrón analiza la frase completa, pero nuestro sistema sólo tiene en cuenta la salida del mismo para la pabra que se encuentra envuelta en el patrón. Finalmente, se selecciona toda la frase analizada por el analizador general y los nodos correspondientes al patrón analizados por el analizador específico.

Primero, hicimos un estudio en profundidad de cada una de las palabras mostradas en la Sección 11.1.1. Este estudio consistió en encontrar los casos diferentes en los que estas palabras se encuentran envueltas en las frases, tanto a la parte correspondiente al etiquetado como a la parte correspondiente a los arcos. Para cada uno de estos casos, seleccionamos las frases del corpus de entrenamiento y entrenamos un analizador específico sólo con esas frases. Como resultado, vimos que la conjunción es la palabra que causó un error más frecuéntemente. Por ello, lo seleccionamos como primer caso de estudio para averiguar si este tipo de analizadores específicos son viables para mejorar la precisión de los analizadores. Los casos estudiados se muestran en la Tabla 11.1.

Como primer experimento, extrajimos automáticamente todas las frases que contenían conjunciones del corpus generando un subconjunto de 1586 frases, con al menos una conjunción. Acto seguido, investigamos los posibles patrones en los que la conjunción está anotada en el corpus. Por ejemplo, encontramos un patrón en el que la conjunción actúa como nexo en frases coordinadas copulativas, y otro patrón cuando actúa como nexo en una lista de nombres. En la siguiente frase: *Los activos en divisas en poder del Banco Central* **y** *el Ministerio de Finanzas se calculan en dólares estadounidenses* **y** *su valor depende del cambio oficial rublo–dólar que establece el Banco Central*, la primera *y* es un nexo entre los nombres propios *Banco Central* y

*Ministerio de Finanzas* y la segunda *y* actúa como un nexo en una coordinada copulativa. Para el resto de palabras llevamos a cabo un estudio similar, mostrado a continuación.

### Aproximación Inicial Semi-Automática

Entrenamos un modelo específico para frases coordinadas copulativas para comprobar la viabilidad de construir n parsers específico para obtener una precisión más elevada. Con este fin, extrajimos un subconjunto del conjunto de entrenamiento con el conjunto de frases coordinadas copulativas, este corpus específico contenía 361 frases (10.561 palabras). Acto seguido hicimos el mismo procedimiento con el corpus de test obteniendo un subconjunto de 16 frases y 549 palabras. Hicimos los experimentos utilizando la misma configuración que utilizó el grupo de Nivre en la CoNLL–X Shared Task. Encontramos que la conjunción se anotó incorrectamente 8 veces (en un conjunto de test con 16 conjunciones). Esto nos llevó a investigar con distintos modelos de características. Después de una serie de intentos, encontramos un modelo de características que utilizado para entrenar alcanzó 12 de 16 conjunciones. A pesar de ello, el modelo general analizó correctamente 13 de las 16 conjunciones. Acto seguido, llevamos a cabo el experimento con el resto de palabras y sus correspondientes patrones.

### Aproximación Semi-Automática Definitiva: N Parsers

Dado que la precisión obtenida por los dos modelos en la primera aproximación fueron muy similares, llevamos a cabo otros experimentos para confirmar o rechazar nuestra hipótesis. Por lo tanto, generamos nuevos analizadores para el resto de palabras con mayor número de errores. Entrenamos un analizador para cada patrón específico que encontramos y para cada palabra concreta. Una vez que la frase es analizada con el modelo específico, el resultado de la palabra "problemática" se intercambia en el árbol producido por el modelo general.

Los resultados obtenidos para todas estas palabras se muestran en la Tabla 11.1. Son consistentemente mejores cuando se utiliza un analizador específico que cuando se utiliza el analizador general por si solo. Pero en algunos casos el analizador general alcanza exactamente los mismos resultados que los analizadores específicos, en estos casos no tendría sentido utilizar los analizadores específicos. Solamente en el caso en el que se analiza la conjunción actuando como nexo en frases coordinadas copulativas, no pudimos encontrar un parser específico mejor que al analizador general. Sin embargo, en 21 de los 28 casos es mejor usar el analizador específico.

En algunos casos la mejora parece muy importante. Por ejemplo, cuando analizamos la palabra *de* si está conectada a un verbo, el parser general muestra un 0% LAS y el parser específico muestra un 100% LAS. Esto es

| Word | | Case | | | | | |
|---|---|---|---|---|---|---|---|
| | | #1 | #2 | #3 | #4 | #5 | #6 |
| y/e | **Label** | – | – | – | – | | |
| | **Attached to a** | verb$^{\leftarrow}$ | proper noun$^{\leftarrow}$ | common noun$^{\leftarrow}$ | adjective$^{\leftarrow}$ | | |
| | **LAS$_{y/e}$ before** | ***81.3***% | 80% | 66.7% | 80% | | |
| | **LAS$_{y/e}$ after** | 75% | ***100***% | ***80***% | ***100***% | | |
| a | **Label** | CD | CI | CC | CREG | – | – |
| | **Attached to a** | verb$^{\leftarrow}$ | | | | | noun$^{\leftarrow}$ |
| | **LAS$_{a}$ before** | 62.5% | 42.9% | 60% | 25% | ***0***% | 50% |
| | **LAS$_{a}$ after** | ***87.5***% | ***100***% | ***100***% | ***75***% | ***0***% | ***100***% |
| de | **Label** | CC | CREG | – | – | | |
| | **Attached to a** | verb$^{\leftarrow}$ | | adverb$^{\leftarrow}$ adjective$^{\leftarrow}$ | noun$^{\leftarrow}$ | | |
| | **LAS$_{de}$ before** | 0% | 0% | ***100***% | 83.3% | | |
| | **LAS$_{de}$ after** | ***100***% | ***100***% | ***100***% | ***96.7***% | | |
| que | **Label** | SUJ | – | SUJ | | | |
| | **Attached to a** | verb$^{\rightarrow}$ | | verb$^{\leftarrow}$ | | | |
| | **LAS$_{que}$ before** | 88.5% | 86.4% | 0% | | | |
| | **LAS$_{que}$ after** | ***92.3***% | ***95.5***% | ***100***% | | | |
| en | **Label** | CC | CC | CREG | – | | |
| | **Attached to a** | verb$^{\rightarrow}$ | verb$^{\leftarrow}$ | | noun$^{\leftarrow}$ | | |
| | **LAS$_{en}$ before** | ***83.3***% | 92.6% | 50% | 62.5% | | |
| | **LAS$_{en}$ after** | ***83.3***% | ***100***% | ***100***% | ***87.5***% | | |
| con | **Label** | CC | CREG | – | – | | |
| | **Attached to a** | verb$^{\leftarrow}$ | | | noun$^{\leftarrow}$ | | |
| | **LAS$_{con}$ before** | 60% | 40% | ***100***% | 66.7% | | |
| | **LAS$_{con}$ after** | ***80***% | ***100***% | ***100***% | ***83.3***% | | |
| por | **Label** | – | CAG | CAG | | | |
| | **Attached to a** | noun$^{\leftarrow}$ | comma$^{\leftarrow}$ | adjective$^{\leftarrow}$ | | | |
| | **LAS$_{por}$ before** | ***100***% | ***100***% | 80% | | | |
| | **LAS$_{por}$ after** | ***100***% | ***100***% | ***100***% | | | |

Table 11.1: Resultados para todas las palabras estudiadas en el corpus de español. LAS específico para cada palabra y cada caso, antes y después de la aplicación de nuestro método.

debido al tamaño del corpus específico, que es muy pequeño. Por ejemplo, si el conjunto de test contiene sólo una aparición para un caso específico y si esta aparición se analiza correctamente, entonces obtenemos un 100% LAS. Sin embargo, el análisis es razonablemente homogéneo y precisiones similares se deberían obtener incluso cuando aumentamos el número de ejemplos en el conjunto de test.

### Resultados del Experimento Semi-Automático

El uso de estos analizadores específicos puede mejorar los resultados. De hecho, si estamos mejorando los resultados de estas palabras tan importantes, estamos generando un efecto muy positivo, ya que como se comentó arriba, estas palabras suelen ser la raíz de subárboles.

Después de analizar el corpus de test, encontramos que nuestro sistema semi-automático alcanza un 20.3% LCM, mientras que el analizador general

alcanzaba un 18.4% LCM. Esta mejora se traduce también en una mejora contabilizada token a token, obteniendo 82.68% LAS, 85.73% UAS y 90.84% LA. Lo que significa una mejora de 1.38% LAS, 1.06% UAS y 0.78% LA si lo comparamos con los resultados del analizador general. La Figura 11.1 muestra estas mejoras contabilizándolos palabra por palabra.

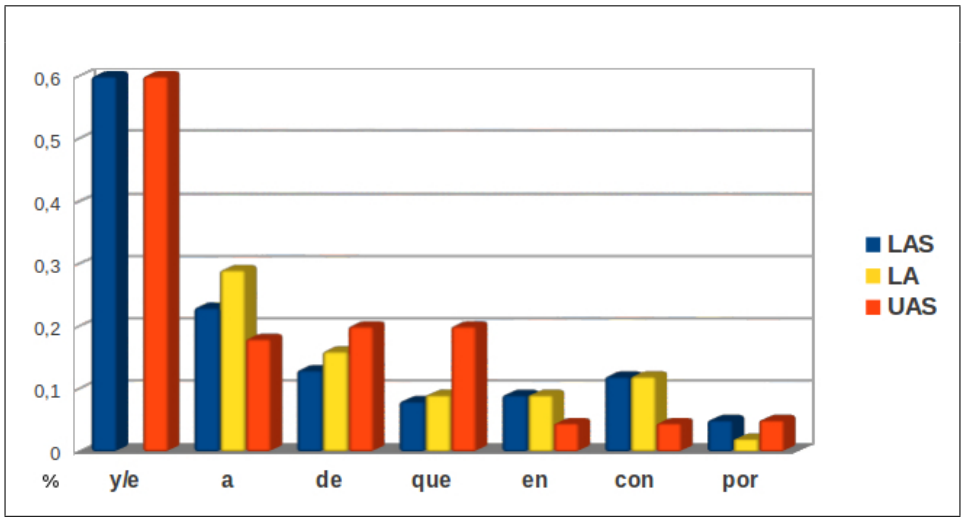



Figure 11.1: Incrementos de LAS, UAS y LA después de la acción de los analizadores específicos.

Como era esperado, las palabras más frecuentes, como la conjunción, afectan en mayor medida el resultado final. Pero en resultados absolutos, todas las palabras alcanzan unos valores similares. Sin embargo, es más útil actuar sobre las palabras más frecuentes, aunque un analizador muy complejo con muchos analizadores específicos, incluyendo incluso palabras no tan frecuentes, podría alcanzar mejores resultados.

### 11.1.3 El Algoritmo del Analizador de N-Versiones

Una vez que nos dimos cuenta que este tipo de analizador combinado es viable, y puede servir para mejorar la precisión, el siguiente paso era hacerlo completamente automático – es decir, el algoritmo que es capaz de hacer trabajar a todos los analizadores especiíficos en sinergia.

El algoritmo que llevamos a cabo está basado en buscar los patrones mediante un sistema de reglas, y enviar las frases a los analizadores más apropiados. Este algoritmo sólo se implementó para la preposición “a” y la conjunción.

El funcionamiento general se muestra en las Figuras 11.2 y 11.3, donde se observa un ejemplo para la preposición “a” y se evidencia como usando los analizadores específicos se puede mejorar la precisión.

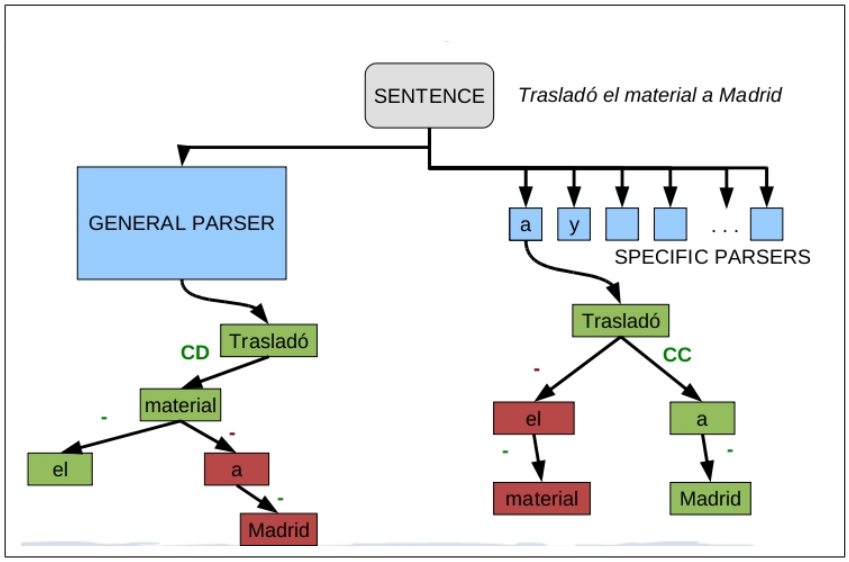



Figure 11.2: El analizador general y el analizador específico analizan la frase: *Trasladó el material a Madrid*.

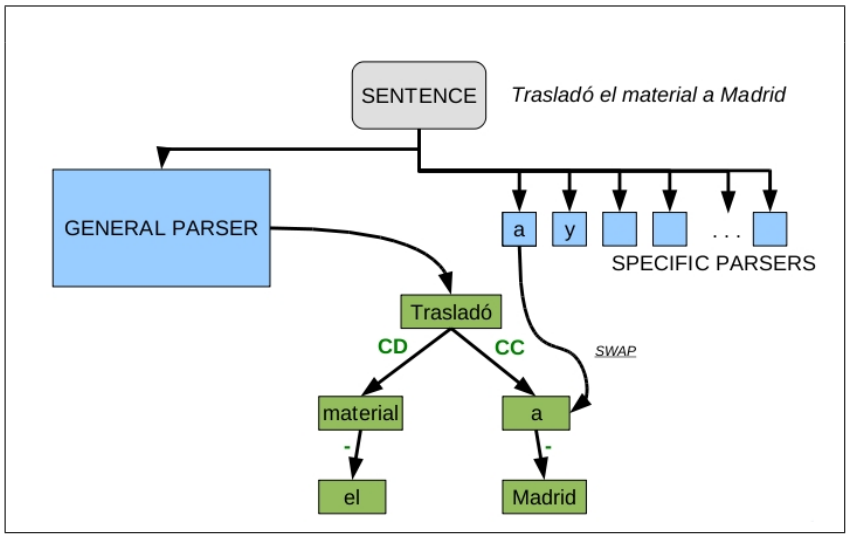



Figure 11.3: Intercambio del nodo que contiene la preposició, generando de ese modo un árbol completamente correcto.

Sin embargo, aplicando este sistema al corpus de español del que disponíamos, vimos que el analizador de n–versiones analiza incorrectamente 55 conjunciones mientras que el analizador general analiza incorrectamente 56 conjunciones. Para la preposición "a" obtuvimos resultados similares, pero en este caso, el analizador general producía mejores resultados (48 errores) que nuestro analizador de n-versiones (50 errores).

Estos resultados no deben llevar a la conclusión inmediata de rechazar este tipo de sistema, ya que averiguamos que en el corpus se encuentran inconsistencias en la anotación de este tipo de palabras, lo que por otro lado hace que el analizador general falle más frecuentemente.

Con lo cual, concluímos que con una validación general del corpus, o de al menos, de estos casos específicos, podríamos observar el funcionamiento de este analizador en un caso más aplicable.

### 11.1.4 Conclusiones

Como primera conclusión y considerando el estudio de viabilidad, parece que merece la pena mejorar la precisión centrándose en palabras concretas que por un lado son relevantes ya que son muy frecuentes, que por otro lado, son la raíz de subárboles y como hemos mencionado, si mejoramos la precisión de estas palabras mejoramos la utilidad de los árboles finales. En la siguiente Sección 11.2 describimos un experimento con resultados muy interesantes en el que nos centramos en la raíz de los árboles de dependencias y lo hacemos de manera totalmente automática. Con lo cual, parece que centrándose en la raíz de los árboles (o subárboles como en este experimento) parece una manera interesante de mejorar la precisión de los analizadores.

Sin embargo, y dado que este experimento sólo es un estudio de viabilidad, concluímos que los corpora deben tener una anotación consistente. No parece aceptable que frases con la misma estructura sintáctica estén anotadas de manera diferente. Con lo cual, y como ya comentamos en las conclusiones del Capítulo anterior, debemos animar al desarrollo de corpora más consistente y donde la selección de frases debe ser muy cuidadosa.

## 11.2 Mejora de un Algoritmo de Parsing Basado en Transiciones Modificando la Posición de la Raíz

Como vimos en la Sección 9.1.1, un algoritmo de parsing basado en transiciones hace uso normalmente de dos estructuras de datos: un buffer y una pila. El buffer contiene todos las unidades léxicas (o palabras) que puestas todas juntas en el orden original conforman la frase original. La pila almacena los nodos que van a ser conectados mediante arcos durante las transiciones del análisis y el algoritmo decide que hacer de acuerdo a lo que encuentra en estas estructuras.

El nodo raíz, que es el que permite tener árboles de dependencias forzando una única raíz para cada frase, normalmente se coloca de manera artificial a la izquierda de la frase, y por lo tanto, se almacena en el primer paso del análisis en la pila. En las publicaciones del estado del arte, la posición de la raíz no es más que una decisión arbitraria que no afecta a la precisión

final y que es simplemente es algo que viene por definición (Nivre, 2006) and (Kübler, McDonald, and Nivre, 2009), ¿Es realmente así?

Hay varios algoritmos de parsing que recorren las frases a analizar de izquierda a derecha y pueden ser agrupados en distintas familias de acuerdo a sus estrategias de análisis. En esta Sección nos centramos principalmente en dos que se comportan de manera distinta: el arc-eager y el arc-standard (Nivre, 2003; Nivre, 2008), ambos están implementados en MaltParser (Nivre, Hall, and Nilsson, 2006). Estos dos algoritmos difieren principalmente en como generan los arcos hacia la izquierda, el algoritmo arc-eager es voraz y lo hace tan pronto como puede, sin embargo el algortimo arc-standard no se comporta de manera voraz. Sabiendo eso, la posición de la raíz parece entonces relevante, ya que lo arcos a la raíz son arcos a la izquierda.

Además, conociendo el hecho de que MaltParser, utilizando el orden de análisis de arc-eager tiende a tener menos precisión en los arcos a la raíz (Nivre, 2008), podemos generar la siguiente hipótesis: la causa de esta situación es la interacción entre la estrategia voraz y la posición de la raíz al principio de la frase, y más importante, la posición de la raíz durante el análisis es relevante y afecta la precisión en distintos escenarios.

En esta Sección, presentamos unos experimentos forzando al analizador a comportarse manera diferente modificando la posición de la raíz.

### 11.2.1 Raíz a la Izquierda versus Raíz a la Derecha

Para corroborar la hipótesis presentada arriba, llevamos a cabo el siguiente experimento: ejecutamos para todos los corpora de la CoNLL-X Shared Task (Buchholz and Marsi, 2006) el algoritmo Nivre arc-eager en los escenarios que se presentan abajo. Los resultados de LAS y UAS se muestran en la Tabla 11.2. Los corpora se ordenan en la Tabla de acuerdo al porcentaje de arcos a la izquierda.

- Raíz a la izquierda: Nivre arc-eager con análisis pseudo-proyectivo y configuraciones por defecto. Por lo tanto, el analizador añade en el primer paso de análisis el nodo raíz a la pila, lo que es lo mismo que tener el nodo al principio (o a la izquierda) de la frase.

- Raíz a la derecha: modificamos los corpora añadiendo una raíz extra al final de las frases. Todos los arcos a la raíz original en los corpora los modificamos y los colocamos a este nodo extra. Después del análisis, procesamos la frase de izquierda a derecha cambiando los arcos al nodo extra al nodo real. El análisis lo hicimos forzando al analizador a ejecutar sin raíz, esto es posible utilizando una opción incluida en MaltParser.

En todos nuestros experimentos proporcionamos resultados con análisis pseudo-proyectivo (Nivre and Nilsson, 2005), con la intención de evitar el ruido producido por los arcos no proyectivos. Con lo que forzamos el analizador a generar los arcos no proyectivos después del análisis.

| Corpora | Raíz-Pos | LAS | UAS |
|---|---|---|---|
| Arabic | Left | 63.71 | 74.53 |
| | Right | **64.15** | **74.97** |
| Danish | Left | 80.74 | 86.71 |
| | Right | **82.38** | **87.94** |
| Bulgarian | Left | 84.64 | 89.81 |
| | Right | **85.76** | **90.78** |
| Spanish | Left | 78.14 | 82.15 |
| | Right | **78.64** | **82.49** |
| Portuguese | Left | 83.71 | 88.36 |
| | Right | **84.17** | **88.62** |
| Swedish | Left | 83.19 | 89.34 |
| | Right | **83.59** | **89.70** |
| Czech | Left | 72.94 | 80.16 |
| | Right | **73.96** | **81.16** |
| German | Left | 83.27 | 86.10 |
| | Right | **83.93** | **86.72** |
| Slovene | Left | 67.75 | 77.84 |
| | Right | **69.98** | **79.62** |
| Dutch | Left | 70.95 | **74.55** |
| | Right | **71.05** | 74.51 |
| Chinese | Left | 84.55 | 89.07 |
| | Right | **85.15** | **89.70** |
| Japanese | Left | 88.71 | 91.25 |
| | Right | **89.77** | **92.12** |
| Turkish | Left | 56.54 | 71.78 |
| | Right | **56.64** | **72.16** |

Table 11.2: Resultados de Nivre arc-eager para los dos escenarios mostrando LAS y UAS.

Como podemos observar en la Tabla 11.2, los resultados porporcionados por el parser cuando el nodo raíz se localiza al final de la frase (a la derecha) produjo una mejora importante en la precisión para la mayoría de los lenguajes. De hecho, produjo mejores resultados para todos los idiomas en LAS y en 12 de los 13 para UAS (la excepción es el holandés). Explicamos estos resultados en las siguientes secciones.

### 11.2.2 Experimento de Control

Con la idea de estar seguros si los resultados obtenidos en la Sección anterior 11.2.1 no fueron una cuestión de suerte, presentamos tres escenarios diferentes donde estudiamos la posición de la raíz y su rol durante el análisis. Realizamos experimentos, modificando los corpora de entrenamiento y uti-

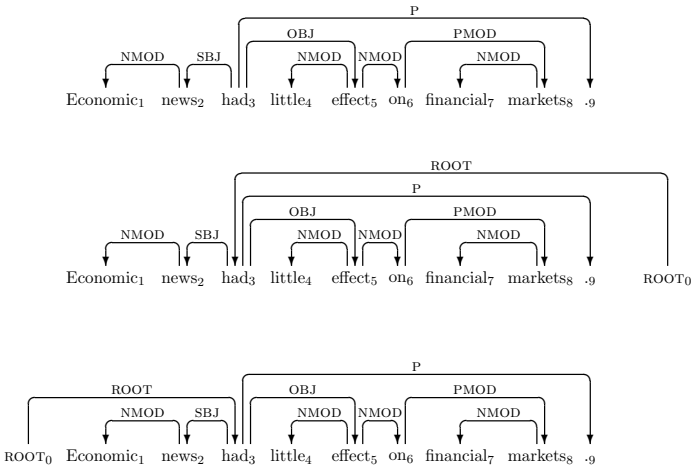



Figure 11.4: Grafos de dependencias de los tipos No raíz (SC-1), Derecha (SC-2) e Izquierda (SC-3) para una frase en inglés extraida del Penn Treebank.

lizando las opciones del tratamiento del nodo raíz proporcionadas en MaltParser.

- Escenario 1 (SC-1): No existencia de raíz durante el análisis. El analizador en la última fase de análisis conecta todos los nodos sin padre a un nodo raíz, con la etiqueta 'ROOT' por defecto. Esto puede hacerse utilizando la opción "allow root" de MaltParser en su configuración *false*, lo que significa que no hay raíz durante las transiciones del análisis.

- Escenario 2 (SC-2): Raíz a la derecha de la frase. Modificamos los corpora añadiendo un nodo raíz extra localizado al final de la frase. Todos los arcos a la raíz se modifican previamente y se conectan a este nodo raíz. Después del análisis, hacemos el paso contrario conectando todos esos arcos a la raíz original.

- Escenario 3 (SC-3): Raíz a la izquierda de la frase. Lo mismo que en SC-2 pero añadiendo el nodo raíz a la izquierda (o al principio) de la frase.

Las Figuras 11.4 y 11.5 ilustran los tres tipos de grafos de dependencias posibles con ejemplos extraídos del corpus Penn Treebank en inglés y del corpus checo.

Para corroborar nuestra hipótesis llevamos a cabo el experimento con el algoritmo Nivre arc-eager y el algoritmo Nivre arc-standard.

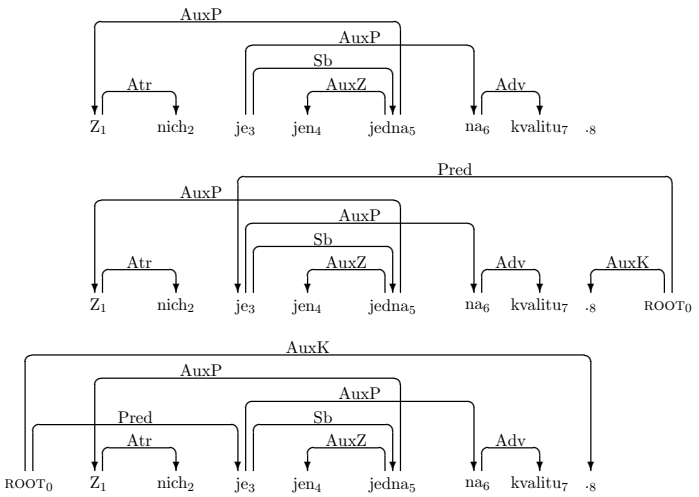



Figure 11.5: Grafos de dependencias de los tipos No raíz (SC-1), Derecha (SC-2) e Izquierda (SC-3) para una frase en checo extraida del Prague Dependency Treebank. La frase significa ("Sólo uno de ellos tiene que ver con la calidad")

### Nivre arc-eager

Para el algoritmo Nivre arc-eager tenemos las siguientes hipótesis:

- Para SC-2 y SC-3, esperamos resultados similares a los producidos en el experimento anterior mostrado en la Sección 11.2.1.
- Para SC-1 esperamos un resultados similar al que tenemos con SC-2, con una perdida importante de precisión en los lenguajes que tienen diversas etiquetas de raíz (árabe, checo, esloveno y portugués) ya que el analizador usará la etiqueta 'ROOT' perdiendo precisión en el etiquetado. Además, esperamos que SC-2 sea un poco mejor ya que con este escenario lo que hacemos es enriquecer la forma de generar arcos a la raíz en el análisis pseudo-proyectivo.

Los resultados del experimento se muestran en la Tabla 11.3.

En SC-3 el nodo raíz está constantemente en uso, mientras que en SC-2 sólo aparece al final. Con lo cual, en este caso el comportamiento voraz de Nivre arc-eager produce errores en SC-3 ya que genera los arcos hacia la izquierda, tan pronto como puede, generando arcos incorrectos a este nodo raíz.

En los corpora con una proporción elevada de arcos a la izquierda, en SC-3 se producen menos arcos a la derecha que en los otros 2 escenarios,

| | | Root Attachments | | | Left Attachments | | | Right Attachments | | | LAS and UAS | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Corpora | Root-Pos | # | Recall | Precision | # | Recall | Precision | # | Recall | Precision | LAS | UAS |
| Arabic | No Root (SC-1) | 315 | 74.24 | 69.52 | 4206 | 95.93 | 94.82 | 469 | 61.52 | 70.58 | 60.00 | **75.17** |
| | Right (SC-2) | 318 | 73.56 | 68.24 | 4199 | 95.81 | 94.86 | 473 | 61.71 | 70.19 | **64.15** | 74.97 |
| | Left (SC-3) | 339 | 84.75 | 73.75 | 4252 | 96.87 | 94.71 | 399 | 57.06 | 76.94 | 63.63 | 74.57 |
| Danish | No Root (SC-1) | 335 | 91.33 | 88.06 | 3715 | 97.63 | 97.42 | 960 | 90.71 | 92.60 | 82.36 | 87.88 |
| | Right (SC-2) | 335 | 91.64 | 88.36 | 3715 | 97.65 | 97.44 | 960 | 90.82 | 92.71 | **82.38** | **87.94** |
| | Left (SC-3) | 341 | 86.69 | 82.11 | 3749 | 97.92 | 96.83 | 920 | 87.45 | 93.15 | 80.60 | 86.59 |
| Bulgarian | No Root (SC-1) | 413 | 94.22 | 90.80 | 3205 | 98.19 | 97.91 | 1395 | 95.00 | 96.63 | **85.76** | **90.80** |
| | Right (SC-2) | 414 | 94.22 | 90.58 | 3204 | 98.15 | 97.91 | 1395 | 95.00 | 96.63 | **85.76** | 90.78 |
| | Left (SC-3) | 410 | 90.20 | 87.56 | 3239 | 98.37 | 97.07 | 1364 | 93.02 | 96.77 | 84.64 | 89.83 |
| Spanish | No Root (SC-1) | 215 | 83.25 | 76.28 | 3057 | 96.66 | 95.75 | 1719 | 92.41 | 94.94 | **78.64** | **82.51** |
| | Right (SC-2) | 216 | 83.76 | 76.39 | 3054 | 96.63 | 95.81 | 1721 | 92.53 | 94.94 | **78.64** | 82.49 |
| | Left (SC-3) | 214 | 79.70 | 73.36 | 3070 | 96.70 | 95.37 | 1707 | 91.90 | 95.08 | 78.14 | 82.15 |
| Portuguese | No Root (SC-1) | 309 | 92.01 | 85.76 | 3022 | 98.40 | 97.88 | 1678 | 96.09 | 98.21 | 79.12 | 88.60 |
| | Right (SC-2) | 309 | 92.01 | 85.76 | 3022 | 98.40 | 97.88 | 1678 | 96.09 | 98.21 | **84.17** | **88.62** |
| | Left (SC-3) | 293 | 87.85 | 86.35 | 3037 | 98.54 | 97.53 | 1679 | 95.98 | 98.03 | 83.77 | 88.36 |
| Swedish | No Root (SC-1) | 403 | 93.32 | 90.07 | 2741 | 96.68 | 96.83 | 1877 | 95.39 | 95.90 | 83.49 | 89.60 |
| | Right (SC-2) | 403 | 93.32 | 90.07 | 2745 | 96.83 | 96.83 | 1873 | 95.39 | 96.10 | **83.59** | **89.70** |
| | Left (SC-3) | 399 | 91.77 | 89.47 | 2753 | 96.72 | 96.44 | 1869 | 94.86 | 95.77 | 83.13 | 89.29 |
| Czech | No Root (SC-1) | 382 | 80.51 | 74.61 | 2561 | 92.21 | 92.03 | 2057 | 90.14 | 91.59 | 68.30 | 81.14 |
| | Right (SC-2) | 380 | 81.07 | 75.53 | 2562 | 92.33 | 92.12 | 2058 | 90.19 | 91.59 | **73.96** | **81.16** |
| | Left (SC-3) | 406 | 83.33 | 72.66 | 2587 | 92.72 | 91.61 | 2007 | 88.28 | 91.93 | 72.98 | 79.96 |
| German | No Root (SC-1) | 397 | 94.68 | 85.14 | 2607 | 95.84 | 92.87 | 2004 | 90.49 | 95.96 | 83.85 | 86.64 |
| | Right (SC-2) | 397 | 94.68 | 85.14 | 2605 | 95.88 | 92.98 | 2006 | 90.64 | 96.01 | **83.93** | **86.72** |
| | Left (SC-3) | 354 | 89.64 | 90.40 | 2625 | 96.08 | 92.46 | 2029 | 90.64 | 94.92 | 83.29 | 86.08 |
| Slovene | No Root (SC-1) | 457 | 73.98 | 63.46 | 2255 | 88.07 | 91.35 | 2292 | 89.71 | 88.96 | 64.25 | 79.56 |
| | Right (SC-2) | 459 | 75.00 | 64.05 | 2250 | 88.03 | 91.51 | 2295 | 89.88 | 89.02 | **69.98** | **79.62** |
| | Left (SC-3) | 435 | 71.94 | 64.83 | 2300 | 88.11 | 89.61 | 2269 | 87.86 | 88.01 | 67.73 | 77.84 |
| Dutch | No Root (SC-1) | 510 | 65.56 | 66.08 | 2339 | 87.11 | 87.00 | 2149 | 87.34 | 87.30 | **71.09** | **74.51** |
| | Right (SC-2) | 507 | 65.76 | 66.67 | 2341 | 87.20 | 87.01 | 2150 | 87.34 | 87.26 | 71.05 | **74.51** |
| | Left (SC-3) | 641 | 72.18 | 57.88 | 2275 | 85.10 | 87.38 | 2082 | 86.73 | 89.48 | 70.81 | 74.41 |
| Chinese | No Root (SC-1) | 915 | 93.63 | 88.42 | 1134 | 88.99 | 91.98 | 2921 | 95.43 | 95.86 | 85.13 | 89.68 |
| | Right (SC-2) | 915 | 93.63 | 88.42 | 1134 | 88.99 | 91.98 | 2921 | 95.43 | 95.86 | **85.15** | **89.70** |
| | Left (SC-3) | 890 | 92.25 | 89.55 | 1160 | 88.91 | 89.83 | 2920 | 95.23 | 95.58 | 84.59 | 89.09 |
| Japanese | No Root (SC-1) | 1020 | 92.74 | 85.20 | 412 | 98.32 | 99.51 | 3571 | 95.92 | 98.01 | **89.85** | 92.10 |
| | Right (SC-2) | 1030 | 92.96 | 84.56 | 412 | 98.32 | 99.51 | 3561 | 95.70 | 98.06 | 89.77 | **92.12** |
| | Left (SC-3) | 926 | 88.15 | 89.20 | 418 | 98.56 | 98.33 | 3659 | 97.18 | 96.91 | 88.79 | 91.27 |
| Turkish | No Root (SC-1) | 645 | 90.29 | 92.25 | 254 | 82.99 | 94.09 | 4122 | 99.04 | 97.89 | **56.66** | **72.18** |
| | Right (SC-2) | 646 | 90.14 | 91.95 | 252 | 82.29 | 94.05 | 4123 | 99.04 | 97.87 | 56.64 | 72.16 |
| | Left (SC-3) | 629 | 88.77 | 93.00 | 245 | 79.51 | 93.47 | 4147 | 99.34 | 97.59 | 56.48 | 71.86 |

Table 11.3: Resultados de Nivre arc-eager para cada uno de los tres escenarios, mostrando arcos a la raíz (root attachments), arcos a la izquierda (left attachments), arcos a la derecha (right attachments), LAS y UAS.

y la razón parece ser la misma, el comportamiento de Nivre arc-eager y la acción que genera el nodo raíz "en juego" durante todo el proceso. Esto hace que el analizador falle en los valores de cobertura (recall). Además, en SC-3, este comportamiento afecta los arcos a la raíz, donde se observa una súper producción de los mismos.

### Nivre arc-standard

Para contrastar los resultados obtenidos con Nivre arc-eager, decidimos llevar a cabo el mismo experimento ejecutando el analizador con el algoritmo Nivre arc-standard. El algoritmo arc-standar no genera los arcos hacia la izquierda tan pronto como puede, con lo cual, teníamos la siguiente hipótesis:

no esperamos las mismas mejoras debido al comportamiento del algoritmo arc-standard, aún así, podemos esperar algunas mejoras colocando el nodo raíz al final de la frase.

Consideramos los tres escenarios tenidos en cuenta en el experimento anterior. Los resultados del experimento se muestran en la Tabla 11.4.

| | | Root Attachments | | | Left Attachments | | | Right Attachments | | | LAS and UAS | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Corpora | Root-Pos | # | Recall | Precision | # | Recall | Precision | # | Recall | Precision | LAS | UAS |
| Arabic | No Root (SC-1) | 299 | 81.69 | 80.60 | 4221 | 96.51 | 95.05 | 470 | 64.50 | 73.83 | 60.48 | 77.29 |
| | Right (SC-2) | 301 | 82.71 | 81.06 | 4216 | 96.46 | 95.11 | 473 | 64.50 | 73.36 | **65.29** | **77.31** |
| | Left (SC-3) | 302 | 83.73 | 81.79 | 4213 | 96.49 | 95.21 | 475 | 64.13 | 72.63 | 64.93 | 77.09 |
| Danish | No Root (SC-1) | 322 | 92.88 | 93.17 | 3734 | 98.14 | 97.43 | 954 | 90.61 | 93.08 | 81.64 | **87.86** |
| | Right (SC-2) | 325 | 92.88 | 92.31 | 3731 | 98.06 | 97.43 | 954 | 90.61 | 93.08 | 81.52 | 87.74 |
| | Left (SC-3) | 322 | 92.88 | 93.17 | 3736 | 98.17 | 97.40 | 952 | 90.51 | 93.17 | 81.66 | **87.86** |
| Bulgarian | No Root (SC-1) | 398 | 91.96 | 91.96 | 3198 | 98.00 | 97.94 | 1417 | 95.42 | 95.55 | 85.06 | **90.33** |
| | Right (SC-2) | 398 | 91.96 | 91.96 | 3198 | 98.00 | 97.94 | 1417 | 95.42 | 95.55 | **85.16** | **90.33** |
| | Left (SC-3) | 398 | 91.71 | 91.71 | 3201 | 98.03 | 97.88 | 1414 | 95.28 | 95.62 | 85.12 | **90.33** |
| Spanish | No Root (SC-1) | 197 | 81.73 | 81.73 | 3084 | 96.57 | 94.81 | 1710 | 91.17 | 94.15 | **77.88** | **81.69** |
| | Right (SC-2) | 196 | 80.71 | 81.12 | 3084 | 96.50 | 94.75 | 1711 | 91.11 | 94.04 | 77.72 | 81.55 |
| | Left (SC-3) | 197 | 81.22 | 81.22 | 3079 | 96.50 | 94.90 | 1715 | 91.34 | 94.05 | 77.64 | 81.51 |
| Portuguese | No Root (SC-1) | 288 | 90.28 | 90.28 | 3043 | 98.50 | 97.31 | 1678 | 95.51 | 97.62 | 78.32 | 87.78 |
| | Right (SC-2) | 288 | 90.62 | 90.62 | 3041 | 98.47 | 97.34 | 1680 | 95.57 | 97.56 | **83.47** | **87.80** |
| | Left (SC-3) | 288 | 90.62 | 90.62 | 3042 | 98.47 | 97.30 | 1679 | 95.51 | 97.56 | 83.45 | 87.82 |
| Swedish | No Root (SC-1) | 391 | 92.03 | 91.56 | 2750 | 97.19 | 97.02 | 1880 | 95.71 | 96.06 | **82.65** | **89.42** |
| | Right (SC-2) | 391 | 91.77 | 91.30 | 2750 | 97.16 | 96.98 | 1880 | 95.71 | 96.06 | 82.65 | 89.36 |
| | Left (SC-3) | 389 | 91.77 | 91.77 | 2752 | 97.16 | 96.91 | 1880 | 95.71 | 96.06 | 82.53 | 89.38 |
| Czech | No Root (SC-1) | 424 | 87.85 | 73.35 | 2540 | 92.72 | 93.31 | 2036 | 90.43 | 92.83 | 68.36 | 81.96 |
| | Right (SC-2) | 421 | 87.01 | 73.16 | 2540 | 92.68 | 93.27 | 2039 | 90.48 | 92.74 | 74.28 | 81.78 |
| | Left (SC-3) | 344 | 86.44 | 88.95 | 2617 | 94.68 | 92.47 | 2039 | 90.53 | 92.79 | **74.88** | **82.52** |
| German | No Root (SC-1) | 357 | 93.84 | 93.84 | 2607 | 96.20 | 93.21 | 2044 | 91.67 | 95.30 | 84.31 | 87.22 |
| | Right (SC-2) | 357 | 93.84 | 93.84 | 2607 | 96.16 | 93.17 | 2044 | 91.62 | 95.25 | 84.35 | 87.22 |
| | Left (SC-3) | 357 | 93.84 | 93.84 | 2598 | 96.00 | 93.84 | 2053 | 91.86 | 95.08 | **84.37** | **87.24** |
| Slovene | No Root (SC-1) | 438 | 75.26 | 67.35 | 2309 | 89.14 | 90.30 | 2257 | 88.87 | 89.50 | 63.67 | 79.28 |
| | Right (SC-2) | 436 | 75.26 | 67.66 | 2326 | 89.53 | 90.03 | 2242 | 88.47 | 89.70 | **69.42** | 79.28 |
| | Left (SC-3) | 366 | 73.47 | 78.69 | 2383 | 90.21 | 88.54 | 2255 | 88.96 | 89.67 | 69.40 | **79.42** |
| Dutch | No Root (SC-1) | 514 | 69.84 | 72.53 | 2366 | 87.33 | 86.22 | 2137 | 87.52 | 87.97 | 70.67 | 74.43 |
| | Right (SC-2) | 496 | 69.84 | 72.38 | 2366 | 87.33 | 86.22 | 2136 | 87.48 | 87.97 | 70.65 | 74.45 |
| | Left (SC-3) | 402 | 64.20 | 82.09 | 2453 | 90.37 | 86.06 | 2143 | 87.57 | 87.77 | **71.07** | **75.23** |
| Chinese | No Root (SC-1) | 864 | 93.06 | 93.06 | 1157 | 90.10 | 91.27 | 2949 | 96.56 | 96.07 | **85.25** | 90.08 |
| | Right (SC-2) | 864 | 92.82 | 92.82 | 1158 | 90.10 | 91.19 | 2948 | 96.52 | 96.07 | 85.17 | 90.00 |
| | Left (SC-3) | 864 | 92.82 | 92.82 | 1156 | 90.10 | 91.35 | 2950 | 96.59 | 96.07 | 85.23 | **90.10** |
| Japanese | No Root (SC-1) | 1015 | 92.53 | 85.42 | 412 | 98.08 | 99.27 | 3576 | 96.03 | 97.99 | **90.15** | **92.30** |
| | Right (SC-2) | 1014 | 92.64 | 85.60 | 412 | 98.08 | 99.27 | 3577 | 96.08 | 98.02 | 90.01 | 92.28 |
| | Left (SC-3) | 905 | 87.83 | 90.94 | 424 | 98.56 | 96.93 | 3674 | 97.70 | 97.03 | 89.13 | 91.57 |
| Turkish | No Root (SC-1) | 648 | 90.59 | 92.13 | 256 | 82.29 | 92.58 | 4117 | 99.02 | 97.98 | **57.00** | 72.06 |
| | Right (SC-2) | 648 | 90.59 | 92.13 | 255 | 82.29 | 92.94 | 4118 | 99.04 | 97.98 | 56.88 | 72.10 |
| | Left (SC-3) | 623 | 89.68 | 94.86 | 272 | 83.33 | 88.24 | 4126 | 99.21 | 97.96 | 56.80 | **72.12** |

Table 11.4: Resultados de Nivre arc-standard para cada uno de los tres escenarios, mostrando arcos a la raíz (root attachments), arcos a la izquierda (left attachments), arcos a la derecha (right attachments), LAS y UAS.

La conclusión principal es basicamente que nuestra hipótesis se verifica, el algoritmo arc-eager produce muchos arcos incorrectos cuando el nodo raíz se situa a la izquierda, sin embargo, esto no pasa con el algoritmo arc-standard. El algoritmo arc-standard es más conservador cuando genera los arcos hacia la izquierda. De hecho, como podemos observar en los resultados experimentales, en la Tabla 11.4 los tres modelos producen resultados muy

similares. Sin embargo, de nuevo los peores resultados son en mayoría para SC-3.

En algunos corpora (checo, esloveno, holandés y japonés) vemos que el número de nodos raíz del escenario SC-3 desciende de manera sustancial, lo que produce una perdida de los valores de cobertura (recall) pero una mejora en la precisión. En estos casos, vemos que en SC-3, el número de arcos hacia la izquierda crece generando los arcos al nodo raíz artificial. De nuevo, parece que a pesar del comportamiento de arc-standard, la presencia del nodo raíz durante el proceso afecta el comportamiento del analizador, y en definitiva, la precisión.

Comparando SC-1 con SC-2, podemos extraer conclusiones similares a las que teníamos con Nivre arc-eager, poniendo el nodo a la derecha de la frase (SC-2), tenemos basicamente el mismo comportamiento que no teniendo raíz (SC-1), pero mejorando el paso final del análisis ya que tenemos más información, por el etiquetado de los nodos raíz.

### 11.2.3 Experimentos con MSTParser

Finalmente, para ver si la posición de la raíz sólo es una cuestión de estudio en los algoritmos basados en transiciones, decidimos ejecutar los mismos experimentos con MSTParser (McDonald, Lerman, and Pereira, 2006), un parser basado en grafos.[1] En esta perspectiva, el algoritmo de análisis no impone ningún orden entre las palabras, con lo cual esperamos menos (o ningún) impacto al modificar la posición de la raíz.

Para poder tener los mismos escenarios que teníamos con arc-eager y arc-standard, modificamos el código de MSTParser, forzando al analizador a dar peso cero a los arcos generados a la raíz implícita en el parser, de ese modo en los tres escenarios los arcos a la raíz se generan con los nodos artificiales (en SC-2 y SC-3), y sin raíz en SC-1.

Como hemos comentado, en este caso esperamos los mismos resultados en los tres escenarios, o basicamente, sólo ruido entre un escenario y otro. Los resultados se muestran en la Tabla 11.5.

Se observa que las diferencias entre los distintos escenarios son más o menos aleatorias. En un algoritmo basado en grafos, como MSTParser, este deber ser el caso, ya que los arcos a la raíz no interfieren en el orden de análisis como es el caso de los algoritmos basados en transiciones.

### 11.2.4 Conclusiones

De este experimento hay dos conclusiones principales que podemos extraer. La primera es que, para algunos modelos de análisis, la existencia y el emplazamiento del nodo raíz es *de facto* un parámetro que se debe tener en

[1] Ver la Sección 9.2

| | | Root Attachments | | | Left Attachments | | | Right Attachments | | | LAS and UAS | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Corpora | Root-Pos | # | Recall | Precision | # | Recall | Precision | # | Recall | Precision | LAS | UAS |
| Arabic | No Root (SC-1) | 296 | 84.07 | 83.78 | 4265 | 97.45 | 94.98 | 429 | 63.20 | 79.25 | **66.73** | **78.96** |
| | Right (SC-2) | 263 | 78.31 | 87.83 | 4284 | 97.64 | 94.75 | 443 | 64.13 | 77.88 | 66.41 | 78.32 |
| | Left (SC-3) | 272 | 83.39 | 90.44 | 4281 | 97.71 | 94.88 | 437 | 63.20 | 77.80 | 66.55 | 78.68 |
| Danish | No Root (SC-1) | 327 | 92.57 | 91.44 | 3713 | 98.17 | 91.44 | 970 | 98.17 | 98.01 | 83.39 | 89.46 |
| | Right (SC-2) | 323 | 92.26 | 92.26 | 3704 | 98.06 | 98.14 | 983 | 93.27 | 92.98 | 83.43 | 89.42 |
| | Left (SC-3) | 324 | 94.74 | 94.44 | 3717 | 98.30 | 98.04 | 969 | 92.65 | 93.70 | **83.97** | **89.84** |
| Bulgarian | No Root (SC-1) | 398 | 98.24 | 98.24 | 3211 | 98.75 | 98.29 | 1404 | 96.34 | 97.36 | 86.30 | **91.64** |
| | Right (SC-2) | 398 | 97.24 | 97.24 | 3208 | 98.56 | 98.19 | 1407 | 96.41 | 97.23 | 86.14 | 91.28 |
| | Left (SC-3) | 398 | 97.49 | 97.49 | 3198 | 98.44 | 98.37 | 1417 | 96.62 | 96.75 | **86.32** | 91.28 |
| Spanish | No Root (SC-1) | 200 | 79.70 | 78.50 | 3036 | 96.00 | 95.75 | 1755 | 93.15 | 93.73 | 79.40 | **83.57** |
| | Right (SC-2) | 200 | 84.77 | 83.50 | 3043 | 96.27 | 95.79 | 1748 | 93.32 | 94.28 | **79.48** | 83.53 |
| | Left (SC-3) | 196 | 83.76 | 84.18 | 3043 | 96.30 | 95.83 | 1752 | 93.26 | 94.01 | 79.20 | 83.41 |
| Portuguese | No Root (SC-1) | 288 | 89.58 | 89.58 | 2997 | 98.00 | 98.30 | 1724 | 97.43 | 96.93 | 84.87 | 89.74 |
| | Right (SC-2) | 289 | 90.62 | 90.31 | 2996 | 98.20 | 98.53 | 1724 | 97.61 | 97.10 | 84.89 | 89.26 |
| | Left (SC-3) | 288 | 91.67 | 91.67 | 2997 | 98.20 | 98.50 | 1724 | 97.61 | 97.10 | **85.19** | **90.26** |
| Swedish | No Root (SC-1) | 388 | 89.97 | 90.21 | 2735 | 96.25 | 96.60 | 1898 | 95.34 | 94.78 | 81.36 | 88.29 |
| | Right (SC-2) | 389 | 92.03 | 92.03 | 2722 | 96.10 | 96.91 | 1910 | 95.87 | 94.71 | 81.66 | 88.35 |
| | Left (SC-3) | 388 | 91.52 | 91.75 | 2723 | 96.25 | 97.03 | 1910 | 96.03 | 94.87 | **81.76** | **88.59** |
| Czech | No Root (SC-1) | 360 | 82.20 | 80.83 | 2592 | 94.64 | 93.33 | 2048 | 91.63 | 93.51 | 76.70 | 85.98 |
| | Right (SC-2) | 358 | 89.55 | 89.55 | 2592 | 94.99 | 93.67 | 2050 | 91.96 | 93.76 | **77.68** | **86.70** |
| | Left (SC-3) | 358 | 85.88 | 84.92 | 2608 | 96.31 | 93.40 | 2034 | 91.58 | 94.10 | 77.04 | 86.34 |
| German | No Root (SC-3) | 357 | 97.76 | 97.76 | 2569 | 96.63 | 95.02 | 2082 | 93.93 | 95.87 | 85.64 | 89.54 |
| | Right (SC-2) | 357 | 97.76 | 97.76 | 2564 | 96.44 | 95.01 | 2087 | 93.93 | 95.64 | 85.34 | 89.50 |
| | Left (SC-3) | 357 | 97.48 | 97.48 | 2563 | 96.52 | 95.12 | 2088 | 94.07 | 95.74 | **85.74** | **89.66** |
| Slovene | No Root (SC-1) | 408 | 79.08 | 75.98 | 2360 | 91.83 | 91.02 | 2236 | 90.37 | 91.86 | 71.44 | 82.47 |
| | Right (SC-2) | 380 | 76.79 | 79.21 | 2332 | 91.19 | 91.47 | 2292 | 91.60 | 90.84 | 71.64 | 82.33 |
| | Left (SC-3) | 392 | 79.34 | 79.34 | 2351 | 91.88 | 91.41 | 2261 | 90.98 | 91.46 | **71.72** | **82.67** |
| Dutch | No Root (SC-1) | 513 | 79.77 | 79.92 | 2347 | 90.97 | 90.54 | 2347 | 90.46 | 90.88 | **79.05** | **83.49** |
| | Right (SC-2) | 453 | 75.10 | 85.21 | 2386 | 91.61 | 89.69 | 2159 | 90.78 | 90.32 | 78.25 | 82.95 |
| | Left (SC-3) | 457 | 74.32 | 83.59 | 2394 | 92.17 | 89.93 | 2147 | 90.88 | 90.92 | 78.91 | 83.43 |
| Chinese | No Root (SC-1) | 864 | 94.33 | 94.33 | 1193 | 92.15 | 90.53 | 2913 | 96.05 | 96.74 | **86.88** | **90.82** |
| | Right (SC-2) | 863 | 93.87 | 93.97 | 1185 | 91.30 | 90.30 | 2922 | 96.01 | 96.41 | 86.36 | 90.52 |
| | Left (SC-3) | 864 | 94.33 | 94.33 | 1202 | 92.15 | 89.85 | 2904 | 95.74 | 96.73 | 86.54 | 90.68 |
| Japanese | No Root (SC-1) | 989 | 93.38 | 88.47 | 412 | 98.32 | 99.51 | 3602 | 96.82 | 98.08 | 90.45 | 93.02 |
| | Right (SC-2) | 953 | 92.21 | 90.66 | 409 | 97.84 | 99.76 | 3641 | 97.56 | 97.78 | 90.47 | 93.06 |
| | Left (SC-3) | 950 | 92.85 | 91.58 | 413 | 98.32 | 99.27 | 3640 | 97.75 | 97.99 | **90.83** | **93.36** |
| Turkish | No Root (SC-1) | 709 | 93.47 | 86.88 | 274 | 86.46 | 90.88 | 4038 | 97.69 | 98.56 | 58.49 | 74.55 |
| | Right (SC-2) | 649 | 93.02 | 94.45 | 273 | 85.42 | 90.11 | 4099 | 99.04 | 98.44 | **58.89** | **74.83** |
| | Left (SC-3) | 647 | 92.56 | 94.28 | 278 | 86.81 | 89.83 | 4096 | 98.94 | 98.41 | 58.59 | 74.59 |

Table 11.5: Resultados de MSTParser para cada uno de los tres escenarios, mostrando arcos a la raíz (root attachments), arcos a la izquierda (left attachments), arcos a la derecha (right attachments), LAS y UAS.

cuenta para obtener la mejor precisión. Como mostramos en los experimentos, para el algoritmo Nivre arc-eager, podemos tener mejores resultados colocando el nodo raíz al final de la frase (o quitándolo por completo) en vez de colocarlo al principio de la frase como es actualmente la norma en análisis de dependencias. La segunda conclusión es que, el nodo raíz parece ser una fuente de variación que debe ser controlada en las evaluaciones experimentales. La practica habitual, que consiste en colocar el nodo raíz a la izquierda de la frase es una manera de asegurar la comparabilidad de los resultados pero dada la arbitrariedad de esta decisión y teniendo en cuenta los resultados de nuestro experimento, parece sin duda interesente estudiar otras representaciones.

## 11.3 Resumen del Capítulo

En este Capítulo, hemos mostrado un estudio de viabilidad y un método satisfactorio para mejorar la precisión de los modelos de MaltParser.

Sobre el estudio para una combinación de analizadores propuesto en la Sección 11.1, creemos que llevar a cabo los experimentos de resolver pequeñas partes del análisis puede producir beneficios en la precisión. Este trabajo, debe ser entendido como una aproximación inicial para resolver este tipo de problemas.

En el segundo experimento, o estudio, mostrado en la Sección 11.2, creemos que puede cambiar la manera de pensar de los investigadores sobre análisis de dependencias, ya que hemos demostrado que la posición del nodo raíz es relevante en los resultados, principalmente los investigadores que están interesados en análisis basados en transiciones.

# Chapter 12

# Optimización del Análisis de Dependencias.

El desarrollo de analizadores precisos para nuevos idiomas puede requerir una profunda optimización, una tarea que no es trivial para los desarrolladores de aplicaciones a los que les puede faltar la competencia y la motivación para llevar a cabo numerosos experimentos. Podemos observar este hecho en varios de los experimentos mostrados en esta tesis, cuando se busca nuevos modelos de características en la Sección 11.1 por ejemplo. Esta es una de las razones por las que quisimos tratar de aportar soluciones para acelerar el problema de la optimización de analizadores y de algún modo, incluso aportar mejores resultados pudiendo hacer uso de toda la información anotada en los corpora.

Para ilustrar la importancia de la optimización en el análisis de dependencias, Hall et al. (2007) mostraron diferencias de más de un 3 por ciento absoluto en LAS entre los modelos por defecto de MaltParser y los modelos manualmente optimizadors para algunos lenguajes de la CoNLL 2007 Shared Task. Merece la pena mencionar que usando los metodos presentados en este capíulo, estas diferencias son aún mayores que aquellas típicamente obtenidas cuando se comparan analizadores sobre los mismos conjuntos de datos.

En este capítulo, demostramos que es posible la obtención automática de un configuración para modelos de MaltParser. Ejecutar el sistema para obtener modelos con las configuraciones por defecto puede llevar (de manera muy probable) a la obtención de resultados muy por debajo de los esperados, pero realizar esa optimización es una tarea compleja.

Por lo tanto, para facilitar la optimización de MaltParser, proponemos un sistema al que hemos llamado MaltOptimizer,[1] que automatiza la búsqueda de parámetros óptimos basándose en un análisis del corpus de entrenamiento. Aunque el sistema no garantiza que la configuración obtenida sea la mejor

[1] http://nil.fdi.ucm.es/maltoptimizer

posible, ya que la búsqueda sobre el espacio crece de manera exponencial, nuestros experimentos indican que invariablemente mejora sobre los modelos por defecto y se acerca (o incluso mejora) los resultados obtenidos mediante optimización manual realizada por expertos.

## 12.1 ¿Qué Necesitamos Optimizar?

MaltParser, como se mencionó en los capítulos anteriores es un generador de parsers basado en transiciones. Cuando queremos optimizar MaltParser para un nuevo idioma o un nuevo dominio, hay tres aspectos fundamentales que deben ser tenidos en cuenta:

1. El algoritmo de análisis.
2. Los modelos de features.
3. El algoritmo de aprendizaje.

Durante las siguientes secciones se describen estos aspectos.

### 12.1.1 Algoritmo de Análisis

Seleccionar un algoritmo de análisis significa seleccionar un sistema basado en transiciones además de una serie de restricciones dentro de ese sistema. MaltParser tiene implementados cuatro grupos de algoritmos basados en transiciones:

- Algoritmos de Nivre (Nivre, 2003; Nivre, 2008).
  - Algoritmo Nivre arc-eager.
  - Algoritmo Nivre arc-standard.
- Algoritmos de Covington (Covington, 2001; Nivre, 2008).
  - Algoritmo Covington.
  - Algoritmo de Covington no-proyectivo.
- Algoritmos de Stack (o pila) (Nivre, 2009; Nivre, Kuhlmann, and Hall, 2009).
  - Algoritmo de Stack proyectivo.
  - Algoritmo de Stack no proyectivo.
- Algoritmos Multiplanar (Gómez-Rodríguez and Nivre, 2010).
  - El algoritmo Planar arc-eager.

– El algoritmo 2-Planar arc-eager.

MaltParser utiliza el algoritmo Nivre arc-eager en su configuración por defecto. Además, tanto el grupo de Covington como el grupo de Stack contienen algoritmos que pueden tratar estructuras no proyectivas, y cualquier algoritmo proyectivo puede utilizarse en combinación con análisis pseudo-proyectivo para recuperar los arcos no proyectivos mediante un proceso posterior (Nivre and Nilsson, 2005).

Los algoritmos incluidos en MaltParser están definidos y descritos en la Sección 9.1.1, aquí sólo mostramos lo que hay que seleccionar para tener una configuración óptima.

### 12.1.2 Modelos de Features

Una de las ventajas de los sistemas basados en transiciones es que es posible generar modelos de features (o de características) basados en historia local para predecir la siguiente transición, para ello MaltParser proporciona un lenguaje de especificación que permite definir los modelos de features. Los features están normalmente definidos para actuar en las estructuras de datos principales de los algoritmos de análisis, lo que normalmente incluye al menos una pila (o *stack*) que contiene los tokens parcialmente procesados y un *buffer* continente de los tokens por procesar. Los valores de los features son normalmente atributos lingüísticos de uno o más tokens basados en la información anotada en formato CoNLL.

1. FORM: Palabra.
2. LEMMA: Lemma.
3. CPOSTAG: Estructura gramatical de grano grueso.
4. POSTAG: Estructura gramatical de grano fino.
5. FEATS: Lista de atributos morfosintácticos (como tiempo verbal, género o número).
6. DEPREL: Relación de dependencia con el padre.

Los modelos de características por defecto (que es lo que se utiliza cuando el sistema se ejecuta con la configuración dada) incluyen los siguientes grupos de features:

1. Una ventana amplia de POSTAG (categoría gramatical) en la pila (stack) y el buffer (habitualmente de longitud 6).
2. Una ventana de FORM features en la pila y el buffer (habitualmente de longitud 3).

3. Un pequeño conjunto de features basados en DEPREL de los tokens más centrales del buffer y de la stack (habitualmente de tamaño 4).

4. Un pequeño conjunto de combinaciones de los features comentados arriba, en particular n-gramas de POSTAG y pares de POSTAG y FORM.

### 12.1.3 Algoritmo de Aprendizaje

MaltParser utiliza dos librerías para el aprendizaje automático: LIBSVM (Chang and Lin, 2001) y LIBLINEAR (Fan et al., 2008). El paquete LIBSVM permite el uso de support vector machines con kernels, lo que facilita la selección de features pero tiene el problema de ser altamente ineficiente durante el entrenamiento como el análisis. El paquete LIBLINEAR sólo permite clasificadores lineales, lo que hace al análisis y al entrenamiento muy rápidos pero pone más demanda en la selección de features. Ambos paquetes contienen un número de algoritmos específicos cada uno con sus hiper-parámetros que se pueden optimizar y ambas librerías proporcionan resultados similares. Para el desarrollo de MaltOptimizer, y en el desarrollo de los experimentos del presente Capítulo, hemos restringido nuestra atención a LIBLINEAR en el interés de la eficiencia. MaltParser utiliza LIBSVM como sistema de aprendizaje por defecto, sin embargo, es muy probable que este hecho cambie en futuras versiones de MaltParser.

En MaltParser, tanto LIBSVM como LIBLINEAR proporcionan resultados en el mismo rango de precisión, o como comentan Prudhvi Kosaraju and Kukkadapu (2010) o Gómez-Rodríguez and Fernández-González (2012), LIBLINEAR podría incluso proporcionar mejores resultados que LIBSVM pero para otros lenguajes LIBSVM es mejor. Sin embargo, LIBLINEAR es siempre más rápido y eficiente.

El sistema que se presenta en este Capítulo, MaltOptimizer, solamente explora LIBLINEAR por razones de eficiencia, por lo tanto no elige el algoritmo de aprendizaje. Sin embargo, MaltOptimizer si hace una búsqueda del parámetro de coste $C$, que como se establece en (Fan et al., 2008) es el único paramtetro para los clasificadores lineales. El parámetro $C$ fortalece el margen entre las diferentes clases y compensa que algunos puntos queden mal clasificados. Un valor alto de $C$ puede producir overfitting (o sobre entrenamiento), generando un modelo con mayor error en el entrenamiento pero menor en el test (Cassel, 2009). Sólo hemos explorado el parámetro $C$ para valores entre 0 y 1, ya que cuando es mayor que 1 puede considerarse como un parámetro de penalización.

## 12.2 Optimización de MaltParser: MaltOptimizer

Para poder hacer todo lo que se presenta en la Sección 12.1, desarrollamos MaltOptimizer. Es una herramienta software programada en Java que implementa una búsqueda inteligente por los parámetros proporcionados por MaltParser basándose en las heurísticas descritas por Nivre and Hall (2010). El sistema toma como entrada un conjunto de entrenamiento consistente en un conjunto de frases anotadas con árboles de dependencias en formato CoNLL. El proceso de optimización tiene tres fases distintas en las que el usuario puede interactuar mediante ficheros de opciones:

1. Validación del corpus de entrenamiento, análisis de los datos y optimización inicial.
2. Selección del algoritmo de análisis.
3. Selección de features y optimización del hiper-parámetro de LIBLINEAR.

MaltOptimizer estima los resultados esperados proporcionando labeled attacments score (LAS).[2] Merece la pena destacar que MaltOptimizer utiliza la medida de evaluación (LAS, con o sin símbolos de puntuación) para seleccionar la configuración optima a través de todo el proceso de optimización generando la compartiva entre un algortimo y otro, o por ejemplo haciendo los tests entre los modelos de features.

### 12.2.1 Fase 1: Análisis de Datos y Optimización Inicial

Durante la Fase 1, MaltOptimizer hace un análisis del corpus de entrenamiento, sugiere la mejor estrategía de validación y lleva a cabo algunas optimizaciones iniciales.

#### Análisis del Corpus de Entrenamiento

MaltOptimizers comienza validando que el corpus tiene un formato correcto, usando el script de validación que se utilizó en la CoNLL-X shared task (*validateFormat.py*[3]). Si el corpus no es valido, de acuerdo con el script de validación, MaltOptimizer aborta el proceso e informa al usuario para que pueda solucionarlo, si es posible. Si el script devuelve warnings (o avisos) que indican posibles inconsistencias, el sistema sigue con el proceso pero informa al usuario de los posibles problemas.

Además MaltOptimizer colecta la siguiente información sobre el corpus de entrenamiento:

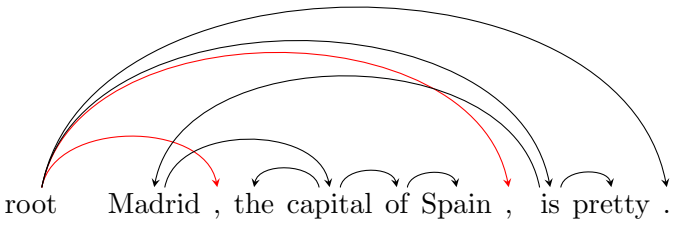
[2]Proporciona LAS en la configuración inicial, pero puede ser modificado proporcionando UAS e incluyendo o excluyendo los signos de puntuación.

[3]http://ilk.uvt.nl/conll/software.html

1. Número de palabras/frases.

2. Porcentaje de arcos y árboles no proyectivos.

3. Existencia de "covered roots" (o raíces cubiertas) (un nodo conectado con la raíz (HEAD = 0) cubierto por un arco que no está conectado con la raíz). El árbol de dependencias, que muestra el árbol con la frase *Madrid, the capital of Spain, is pretty*, mostrado abajo tiene dos covered roots: (i) el que va desde la raíz a la primera coma está cubierto por el arco entre *Madrid* y *capital* que no está directamente conectado con la raíz. (ii) El que va desde la raíz hasta la segunda coma está cubierto por un arco entre *is* y *Madrid* que no está directamente conectado con la raíz.




4. Frecuencia de etiquetas usadas para los tokens conectados a la raíz. Algunos conjuntos de datos utilizan etiquetas distintas (diferentes de *ROOT*), como el checo, portugués, árabe o esloveno.

5. Existencia de features no vaciós en LEMMA y FEATS.

6. Diferencia entre POSTAG y CPOSTAG en el corpus de entrenamiento.

Esta información se va a utilizar durante el resto de pasos de optimización.

### Estrategia de Validación

Basándose en el tamaño del corpus, MaltOptimizer recomienda al usuario elegir entre 2 métodos de validación para usar durante las fases 2 y 3: división simple y cross-validation.

1. División simple (80% para entrenamiento, 20% para test). Se recomienda para conjuntos de datos de más de 90.000 palabras, donde el cross-validation consumiría demasiado tiempo y una división simple es suficientemente grande para dar estimaciones creíbles.

2. 5-fold cross validation (20% en cada división). Cross-validation se recomienda para conjuntos de datos más pequeños, donde una división simple podría ser no suficiente y el tiempo requerido es tolerable.

En ambos casos el sistema selecciona las frases de manera estratificada, para asegurar una distribución similar de los datos en todos los subconjuntos. En todo caso, el usuario puede no utilizar la recomendación del sistema.

### Optimización Inicial

Finalmente, MaltOptimizer hace algunos tests iniciales con el algoritmo por defecto, que como se menciona arriba, es Nivre arc-eager:

- Si hay covered roots, MaltOptimizer prueba las opciones del flag `-pcr`, que puede ser *none* (default), *left*, *right* y *head*. Esto significa que MaltParser conectará los covered roots hacia donde se indica en el flag.

- Si hay más de una etiqueta para los tokens conectados a la raíz (HEAD = 0), MaltOptimizer realiza un test simple para cada etiqueta y de ese modo se puede decidir cual de ellas proporciona mejores resultados. Este test se hace modificando el flag `-grl` proporcionado por MaltParser.

Cuando el análisis del corpus y la optimización inicial se han completado, MaltOptimizer crea un fichero de opciones básico y un fichero de log que se usarán como punto inicial para la optimización. El usuario tiene la oportunidad de editar el fichero de opciones o puede elegir parar el proceso y continuar con la optimización manual, por ejemplo, modificando la etiqueta preferida para los tokens conectados al root. La Figura 12.1 muestra un ejemplo de fichero de opciones y de fichero de log después de la fase 1.

```
phase1_logfile.txt

Training set path:slovene_sdt_train.conll
Size (tokens):28750
Size (sentences):1534
Non projective:22.164276401564535
Dangling Punctuation:27
LAS:66.15
Default:65.63
NumRootLabels:11
JavaHeap:-Xmx2048m
MaxTokens:2147483647
CposEqPos:false
LemmaBlank:false
FeatsBlank:false

---------------------

phase1_optfile.txt

1. root_label(-grl):Pred
2. covered_root(-pcr):right
```

Figure 12.1: Fichero de opciones y fichero de log después de la fase 1.

### 12.2.2 Fase 2: Selección del Algoritmo de Análisis

En la segunda fase, MaltOptimizer explora los algoritmos de análisis implementados en MaltParser, basándose en los resultados de la fase 1 y después, modifica las opciones específicas de cada algoritmo cuando las hay.

#### Algoritmo de Análisis

MaltOptimizer explora los algoritmos de análisis implementados en MaltParser:

- Si no hay árboles/arcos no proyectivos en el corpus de entrenamiento, entonces sólo se exploran algoritmos proyectivos:
  - Nivre arc-eager y Nivre arc-standard (Nivre, 2003; Nivre, 2004).
  - Algoritmo de Covington proyectivo (Covington, 2001; Nivre, 2008).
  - Algoritmo de Stack proyectivo (Nivre, 2009).
  - Algoritmo Planar arc-eager (Gómez-Rodríguez and Nivre, 2010).
- Si el corpus de entrenamiento contiene más del 15% de frases con al menos un arco no proyectivo entonces MaltOptimizer explora los siguientes algoritmos:
  - El algoritmo no proyectivo de Covington (Covington, 2001; Nivre, 2008).
  - Algoritmo de Stack no proyectivo (Nivre, 2009; Nivre, Kuhlmann, and Hall, 2009)
  - Algorimo 2-Planar arc-eager (Gómez-Rodríguez and Nivre, 2010).
  - Cualquier algoritmo proyectivo ejecutado en combinación con análisis pseudo-proyectivo (Nivre and Nilsson, 2005).
- En otro caso, ambos grupos de algoritmos se exploran y MaltOptimizer selecciona el mejor entre todos.

Para reducir el número de tests llevados a cabo, desarrollamos dos árboles de decisión basados en experiencia previa (Nivre and Hall, 2010). El primero, mostrado en la Figura 12.2, comprueba solamente algoritmos proyectivos de manera que el número máximo de tests es 4, y el árbol evita realizar pruebas innecesarias como comprobar el algoritmo Nivre arc-standard cuando el algoritmo Nivre arc-eager proporciona mejores resultados que el algoritmo Stack proyectivo debido a que siguen el mismo orden de análisis. El algoritmo Nivre arc-standard usa el mismo orden de análisis que el algoritmo Stack proyectivo. Además, Planar arc-eager es similar a Nivre arc-eager teniendo en cuenta el orden de análisis y por ello se situa en la misma rama que Nivre arc-eager.

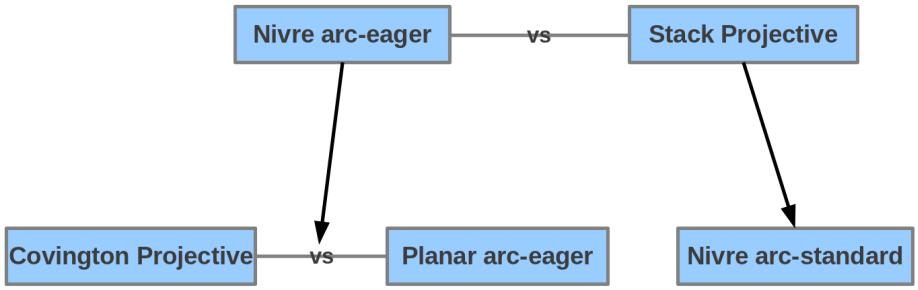



Figure 12.2: Árbol de decisión para seleccionar el mejor algoritmo proyectivo.

El segundo árbol de decisión mostrado en la Figura 12.3, es para algoritmos no proyectivos y resulta en un máximo de 6 tests usando consideraciones similares. Sin embargo, en este caso tenemos un número mayor de algoritmos para considerar.

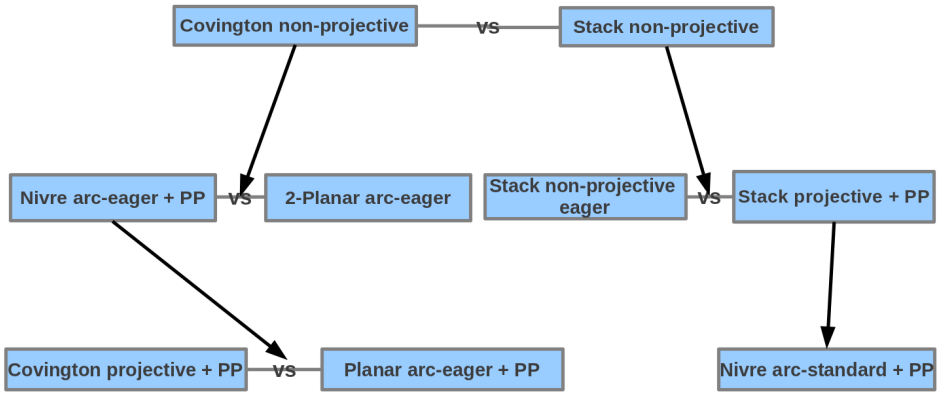



Figure 12.3: Árbol de decisión para seleccionar el mejor algoritmo no proyectivo.

### Optimización de Parámetros

Después de procesar ambos árboles de decisión con las configuraciones por defecto, MaltOptimizer modifica los parámetros del mejor algoritmo y crea un nuevo fichero de opciones para la mejor configuración, siguiendo como criterio los mejores resultados dados por la medida de evaluación. Aquí mostramos los tests que MaltOptimizer realiza para cada algoritmo cuando hay opciones específicas disponibles.

- **Algoritmos de Nivre**: MaltOptimizer comprueba la manera de tratar la raíz de los árboles, llevando a cabo una serie de tests con

las opciones booleanas `allow_root` y `allow_reduce` incluidas en MaltParser.

  - `allow_root`=true (por defecto) significa que el parser usa la raíz como un token más durante el análisis. En otro caso, no hay root durante el análisis.
  - `allow_reduce`=false (por defecto) significa que la transición *reduce* no está permitida si el nodo que está en la cima de la pila es la raíz. En otro caso, se permite y será conectada con la raíz en el último paso del análisis.

- **Algoritmos de Covington**: MaltOptimizer explora de nuevo la estrategia de tratamiento de la raíz, explorando la opción `allow_root`, y la opción `allow_shift`.

  - `allow_root`=true (por defecto) significa que el analizador utiliza la raíz como otro token durante el análisis. En otro caso, no hay raíz durante el análisis.
  - `allow_shift`=false (por defecto) significa que la transición *Shift* no está permitida. En otro caso, está permitida.

- **2-Planar arc-eager**: MaltOptimizer explora las opciones `reduceonswitch` (booleana) y las opciones de tratamiento de la raíz de planar `planar_root_handling`.

  - `reduceonswitch`=false (por defecto) significa que el analizador no reduce cuando hace la transición *switch*. En otro caso, reduce.
  - `planar_root_handling`=normal (por defecto) significa que los hijos de la raíz son conectados mediante transiciones a la derecha. `planar_root_handling`=relaxed, los nodos raíz no conectados durante el análisis son conectados con la etiqueta por defecto como en `allow_root`=false para los algoritmos de Nivre.

- **Algoritmo Pseudo-Proyectivo**: si MaltOptimizer decide que el mejor algoritmo es un algoritmo proyectivo ejecutado en combinación con análisis pseudo-proyectivo, entonces se comprueba las opciones del flag `pp`.

Al final de la fase 2, el usuario tiene la opción de editar el fichero de opciones (o parar el proceso) antes de que la optimización siga adelante. Por ejemplo, el usuario puede cambiar el algoritmo seleccionado o alguna de las opciones mostradas arriba seleccionada por MaltOptimizer como óptima. La Figura 12.4 muestra un ejemplo de fichero de opciones y fichero de log después de la fase 2.

```
phase2_logfile.txt

Training set path:slovene_sdt_train.conll
Size (tokens):28750
Size (sentences):1534
Non projective:22.164276401564535
Dangling Punctuation:27
LAS:68.34
Default:65.63
NumRootLabels:11
JavaHeap:-Xmx2048m
MaxTokens:2147483647
CposEqPos:false
LemmaBlank:false
FeatsBlank:false

---------------------

phase2_optfile.txt

1. root_label (-grl):Pred
2. covered_root (-pcr):right
3. parsing_algorithm (-a):stacklazy
```

Figure 12.4: Ficheros de opciones y de log después de la Fase 2.

### 12.2.3 Fase 3: Selección de Features

En la tercera fase, MaltOptimizer optimiza el modelo de características (o features) dados los parámetros obtenidos durante las fases anteriores, en particular el algoritmo seleccionado. Primero lleva a cabo una selección backward (o hacia atrás) que consiste en borrar features. Y luego realiza una selección forward (hacia delante) probando features potenciales uno a uno.

**Selección Backward** puede ser presentado de manera formal como sigue:

$X = X_1, \ldots, X_n$ es un conjunto de features y M($X$) es la medida de evaluación para $X$ (en nuestro caso, LAS o UAS).

*Mientras* $|X| < 1$
  $B = 0$
  *Mejor = NIL*
  *Para cada* $X_i \in X$
    *Si M(X -* $X_i$*)* $> B$ *entonces*
      $B$ = *M(X -* $X_i$*)*
      *Mejor* = $X_i$
  *Si* $B <$ *M(X) entonces*
    *devolver X*
  *sino*
    $X = X$ - $X_i$
*Devolver X*

Como se observa el número de posibilidades crece exponencialemente, haciendo imposible una búsqueda exhaustiva de features incluso para valores moderados de $X$.

**Selección Forward**. Normalmente, empezar sin variables y añadir los posibles features uno a uno, en cada paso añadir los que funcionan mejor de manera significativa.

Puede presentarse de manera formal como sigue:

X = $X_1, \ldots, X_n$ es el conjunto de features potenciales y M(X) la medida de evaluación para $X$ (en nuestro caso, LAS o UAS).

Let Y = { }
*Mientras* $Y \neq X$
  $B = 0$
  *Mejor = NIL*
  *Para cada* $X_i \in X$
    *Si M(Y U* $X_i$*) >B entonces*
      $B$ = *M(Y U* $X_i$*)*
      *Mejor* = $X_i$
  *Si* $B <$ *M(X) entonces*
    *Devolver Y*
  *sino*
    $Y = Y \ U \ X_i$
    $X = X$ - $X_i$
*Devolver Y*

Una búsqueda exhaustiva es de nuevo prácticamente imposible, por ello nuestra estrategia de optimización está basada en heurísticas derivadas de experiencia previa (Nivre and Hall, 2010). Los pasos principales de nuestra selección forward y selección backward son los siguientes:

1. Modificar la ventana de POSTAG en la pila y el buffer.

2. Modificar la ventana de features léxicos (FORM) en la pila y el buffer.

3. Modificar los features del árbol de dependencias usando DEPREL y POSTAG features.

4. Añadir predecesores y sucesores de features para tokens salientes usando POSTAG y FORM features.

5. Añadir CPOSTAG, FEATS y LEMMA features si están disponibles.

6. Añadir conjunciones de POSTAG y FORM features.

Nuestro algoritmo recorre todos los pasos añadiendo un feature en cada iteración y manteniendo el conjunto de features que produce mejores resultados hasta el momento de manera voraz. Aceleramos el proceso aplicando heurísticas derivadas de experiencia previa, son las siguientes:

1. Si la selección backward produce mejoras para una ventana específica, no se intenta selección forward para esa ventana.

2. Siempre que la selección forward no tiene éxito para una ventana, no intentamos más experimentos forward para esa ventana.

Estos seis pasos son algo diferentes dependiendo de que algoritmo es el mejor con la configuración por defecto, ya que los algoritmos de MaltParser tienen diferente forma de generar las transiciones y utilizan diferentes estructuras de datos, pero los pasos son similares a un cierto nivel de abstracción.

Una vez que la selección de features ha terminado, MaltOptimizer crea un fichero de opciones y un nuevo fichero que contiene el modelo de features. El usuario tiene la oportunidad de editar ambos ficheros.

#### Optimización del Parámetro de LIBLINEAR

Al final de la tercera fase, MaltOptimizer modifica el parámetro $C$ de LIBLINEAR usando una búsqueda voraz simple, empezando en $C$=0.01, iterando diez veces hasta $C$=1.0, seleccionando el valor óptimo de $C$, y este es el que muestra un valor más alto de LAS (o UAS). Cuando finaliza esta optimización, MaltOptimizer crea un fichero de opciones y un fichero de log. El usuario puede ahora continuar y hacer optimización manual. La Figura 12.5 muestra un ejemplo de un fichero de opción y fichero de log después de la fase 3.

## 12.3 Experimentos con MaltOptimizer

Con la intención de comprobar la capacidad de MaltOptimizer, ejecutamos las tres fases en todos los corpora de de la CoNLL 2006 y 2007 Shared Tasks (Buchholz and Marsi, 2006; Nivre et al., 2007). Usamos 5-fold cross-validation para los corpora que tienen menos de 90.000 palabras y división simple para los corpora más grandes siguiendo las sugerencias incluidas en MaltOptimizer.

La Tabla 12.1 muestra los resultados (LAS) mostrando los resultados por defecto y después de ejecutar cada una de las tres fases. Las dos últimas columnas comparan la precisión obtenida con el conjunto final de test (Test-MO) con el mejor resultado obtenido mediante optimización manual con MaltParser (Test-MP) en las shared tasks (Nivre et al., 2006a; Hall et al., 2007). La Tabla 12.2 muestra el algoritmo seleccionado para cada corpus.

```
phase3_logfile.txt

Training set path:slovene_sdt_train.conll
Size (tokens):28750
Size (sentences):1534
Non projective:22.164276401564535
Dangling Punctuation:27
LAS:71.65
Default:65.63
NumRootLabels:11
JavaHeap:-Xmx2048m
MaxTokens:2147483647
CposEqPos:false
LemmaBlank:false
FeatsBlank:false

---------------------

phase3_optfile.txt

1. root_label (-grl):Pred
2. covered_root (-pcr):right
3. parsing_algorithm (-a):stacklazy
8. feature model (-F):addMergPOSTAGS0FORMStack1.xml
```

Figure 12.5: Fichero de opción y fichero de log después de ejecutar la Fase 3.

La primera conclusión que podemos sacar de los resultados es que la optimización mejora la precisión para todos los corpora sin excepción, aunque la mejora varía considerablemente desde 1 punto percentual para chino, japonés y sueco, hasta 7–10 puntos para vasco, holandés, checo, húngaro, y turco. Algunos corpora contienen información que se beneficia de tener features basados en información morfosintáctica y por ello se obtienen mejores resultados con unos que con otros.

Esta también es la razón por la que las mejoras más significativas se producen en la fase 3. Sin embargo, también se observan importantes avances en la fase 2 para los idiomas con una cantidad elevada de dependencias no proyectivas, como el checo, holandés o esloveno, donde la selección del algoritmo de análisis es bastante importante (mirar la Tabla 12.2). En estos casos, se selecciona con frecuencia un algoritmo no proyectivo y estos aportan buenos resultados bajo estas características.

Podemos observar también que MaltOptimizer es competitivo comparado con la versión manualmente optimizada de MaltParser, con una diferencia media de 0.61 y una diferencia máxima negativa de 1.87 (para Catalan 2007). Además, vemos que en 5 de los 23 casos, MaltOptimizer mejora los resultados antiguos, y en algunos casos con importantes mejoras. De hecho, en la CoNLL-X shared task, MaltOptimizer habría acabado tercero con estos resultados, sólo superado por MSTParser (McDonald, Lerman, and Pereira, 2006) y MaltParser (Nivre et al., 2006a).

El tiempo necesario para ejecutar la optimización depende principalmente del tamaño del conjunto de datos y el método de validación usado. Con división simple, requiere más o menos media hora para corpora

| CoNLL-X Shared Task | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Lenguaje** | **Defecto** | **Fase 1** | **Fase 2** | **Fase 3** | **Dif** | **Test-MO** | **Test-MP** |
| Arabic* | 63.02 | 63.03 | 64.03 | 66.37 | 3.35 | 66.20 | **66.71** |
| Bulgarian | 83.19 | 83.19 | 84.00 | 86.03 | 2.84 | 86.44 | **87.41** |
| Chinese | 84.14 | 84.14 | 84.95 | 84.95 | 0.81 | 85.49 | **86.92** |
| Czech | 69.94 | 70.14 | 72.44 | 78.04 | 8.10 | **80.46** | 78.42 |
| Danish | 81.01 | 81.01 | 81.34 | 83.86 | 2.85 | 83.41 | **84.77** |
| Dutch | 74.77 | 74.77 | 78.02 | 82.63 | 7.86 | 77.23 | **78.59** |
| German | 82.36 | 82.36 | 83.56 | 85.91 | 3.55 | 85.24 | **85.82** |
| Japanese | 89.70 | 89.70 | 90.92 | 90.92 | 1.22 | 90.39 | **91.65** |
| Portuguese | 84.11 | 84.31 | 84.75 | 86.52 | 2.41 | 85.85 | **87.60** |
| Slovene* | 66.08 | 66.52 | 67.86 | 72.29 | 6.21 | **73.66** | 70.30 |
| Spanish* | 76.45 | 76.45 | 76.64 | 79.65 | 3.20 | 80.18 | **81.29** |
| Swedish | 83.34 | 83.34 | 83.50 | 84.09 | 0.75 | 83.81 | **84.58** |
| Turkish* | 57.79 | 57.79 | 58.33 | 67.11 | 9.32 | 64.85 | **65.68** |
| **CoNLL 2007 Shared Task** | | | | | | | |
| **Lenguaje** | **Defecto** | **Fase 1** | **Fase 2** | **Fase 3** | **Dif** | **Test-MO** | **Test-MP** |
| Arabic | 67.71 | 67.75 | 67.75 | 70.77 | 3.06 | 73.22 | **74.75** |
| Basque* | 67.69 | 67.83 | 68.29 | 75.05 | 7.36 | 72.19 | **74.99** |
| Catalan | 83.07 | 83.07 | 83.13 | 84.89 | 1.82 | 85.87 | **87.74** |
| Chinese | 84.04 | 84.04 | 85.03 | 86.21 | 2.17 | 82.58 | **83.51** |
| Czech | 70.25 | 70.51 | 72.49 | 77.71 | 7.46 | **78.03** | 77.22 |
| English | 83.84 | 83.84 | 85.34 | 86.61 | 2.77 | 85.17 | 85.81 |
| Greek* | 71.01 | 71.09 | 72.41 | 75.12 | 4.11 | **74.50** | 74.21 |
| Hungarian | 66.42 | 66.42 | 68.21 | 76.53 | 10.11 | 77.17 | **78.09** |
| Italian* | 79.07 | 79.07 | 79.45 | 81.53 | 2.46 | **82.79** | 82.48 |
| Turkish* | 67.45 | 68.38 | 70.67 | 76.91 | 9.46 | 78.93 | **79.24** |

Table 12.1: LAS por fase comparado con los resultados por defecto para todos los corpora de las CoNLL shared tasks. Los lenguajes marcados con * tienen corpora pequeños de menos de 90.000 palabras y han sido optimizados usando 5-fold cross validation; el resto utilizan división simple. Las últimas dos columnas usan muestran los resultados con los conjuntos finales de test obtenidos por MaltOptimizer (Test-MO) comparado con los mejores resultados de MaltParser en las shared tasks (Test-MP)

pequeños, y más o menos un día para corpora grandes. Con 5-fold cross-validation, requiere básicamente cinco veces más tiempo.[4]

Finalmente, merece la pena destacar que los experimentos no muestran mejoras por la modificación del parámetro $C$ de LIBLINEAR, el cual es consistentemente mejor con el valor por defecto para MaltParser (0.1) , sin embargo, por razones de completitud merece la pena explorarlo durante la fase 3, ya que con corpora más pequeños podría producir mejoras.

[4]El tiempo de ejecución depende también del ordenador donde se ejecuten los experimentos.

| CoNLL-X Shared Task | |
|---|---|
| **Corpora** | **Algorithm** |
| Arabic | Stack non-projective |
| Bulgarian | Stack non-projective |
| Chinese | Covington projective |
| Czech | Stack non-projective |
| Danish | Stack non-projective |
| Dutch | Nivre arc-eager + PP |
| German | Covington non-projective |
| Japanese | Covington projective |
| Portuguese | Stack non-projective |
| Slovene | Stack non-projective |
| Spanish | Nivre arc-eager |
| Swedish | Nivre arc-eager + PP |
| Turkish | Covington non-projective |
| **CoNLL 2007 Shared Task** | |
| **Corpora** | **Algorithm** |
| Arabic | Nivre arc-eager + PP |
| Basque | Covington non-projective |
| Catalan | Nivre arc-eager |
| Chinese | Stack projective |
| Czech | Stack projective + PP |
| English | Nivre arc-standard + PP |
| Greek | Stack non-projective |
| Hungarian | Stack projective + PP |
| Italian | Nivre arc-eager + PP |
| Turkish | Covington non-projective |

Table 12.2: Algoritmos seleccionados por MaltOptimizer después de ejecutar la Fase 2. PP significa análisis pseudo-proyectivo (Nivre and Nilsson, 2005).

## 12.4 Conclusiones

Creemos posible extender estas técnicas a diferentes analizadores que usen una especificación similar de los features. En el caso de los analizadores basados en transiciones podría llevarse a cabo transformando el lenguaje de especificación. En otros analizadores, como los basados en grafos, los features son simples ya que no debemos preocuparnos sobre las estructuras de datos presentes en los analizadores basados en transiciones pero podríamos considerar el mismo tipo de features.

Hemos tratado de alterar el orden de los experimentos entre las diferentes

fases, mostradas en la Sección 12.2.3, pero no conseguimos nunca mejora ni diferencias significativas entre los diferentes corpora y los distintos algoritmos. Los resultados no fueron los mismos, pero no existe, hasta ahora, un orden de los experimentos que sea consistentemente mejor que el resto.

Merece la pena remarcar que nuestros métodos pueden ser aplicados a todos los algoritmos de MaltParser, con lo que creemos que podrían ser una forma universal de modificar y optimizar la selección de features en los analizadores basados en transiciones.

## 12.5 Resumen del Capítulo

En este Capítulo hemos presentado MaltOptimizer, un sistema de optimización para MaltParser que puede ayudar a los desarrolladores a adaptar el sistema a nuevos idiomas. Hemos demostrado que usando MaltOptimizer es posible conseguir mejoras sustanciosas sobre las configuraciones por defecto.

Además de los desarrolladores de aplicaciones, MaltOptimizer debería ser útil para investigadores que usen MaltParser como punto de partida o comparativa inicial para sus propios sistemas, de este modo, lo harían con configuración óptima y no con los resultados por defecto, que pueden ser realmente inferiores.

# Chapter 13

# Aplicación del Análisis de Dependencias.

En este Capítulo presentamos algunos trabajos de aplicación donde se evidencia la utilidad del análisis de dependencias para resolver algunos problemas de PLN. Mostramos dos trabajos diferentes, donde aplicamos estructuras de dependencias para propósitos diferentes:

- En la Sección 13.1 mostramos un sistema simple capaz de simplificar frases basándose en la estructura de dependencias.
- En la Sección 13.2 mostramos un sistema que es capaz de inferir el ámbito de señales de negación.

## 13.1 Simplificación de Textos para el Español Usando Análisis de Dependencias

En esta Sección investigamos la tarea de simplificar textos para el español utilizando estructuras sintácticas de dependencias. Nuestra motivación principal fue facilitar la accesibilidad a la información para la gente con problemas cognitivos. Quisimos también demostrar que el análisis de dependencias puede ser muy útil a la hora de alcanzar este objetivo. Con lo cual, este estudio consiste en un primer paso para construir un sistema capaz de simplificar texto en español.

Hay muchas personas con problemas para comprender textos de uso común. Estas personas sólo son capaces de encontrar información explícita en textos cortos haciendo simples inferencias. Algunos estudios[1] que miden la capacidad lectora de la población, muestran que en España, el 30% de la población tiene dificultad para entender textos más allá de un cierto nivel de complejidad.

[1] *http://www.facillectura.es*

Las frases largas, frases coordinadas, claúsulas anidadas, frases en voz pasiva, orden de las palabras que difiere del canónico, y el uso de palabras poco frecuentes en el lenguaje coloquial, aumentan la complejidad en el lenguaje (Siddharthan, 2002), (Klebanov, Knight, and Marcu, 2004), (Devlin and Unthank, 2006), (Caseli et al., 2009).

Hay distintas iniciativas en las que se pueden encontrar algunas ideas de como hacer los textos más fáciles de comprender: Plain Language[2],"European Guidelines for the Production of Easy-to-Read Information"[3] o "Web Content Accessibility Guidelines"[4]. En principio, estas recomendaciones pueden aplicarse a cualquier lengua.

La simplificación de textos puede ser a varios niveles: (i) léxico, sustituyendo palabras por sinónimos más usuales, o (ii) sintáctico, modificando la estructura sintáctica de las frases (Siddharthan, 2003; Max, 2006) o cortando partes de las frases (Petersen and Ostendorf, 2007). Como resultado, se espera que el texto pueda ser más comprensible.

En esta Sección nos centramos en la estructura sintáctica de los textos para maximizar la comprensión lectura de los textos escritos llevando a cabo un experimento simple.

### 13.1.1 Trabajo Relacionado sobre Simplificación de Textos

Los sistemas existentes para simplificación de textos pueden dividirse por: el tipo de sistema (basado en reglas, o machine learning), el tipo de conocimiento usado, y los objetivos del sistema.

Hay una serie de sistemas que están basados en reglas como (Chandrasekar, Doran, and Srinivas, 1996) o (Siddharthan, 2003). Estos sistemas contienen un conjunto de reglas generadas manualmente que se aplican a cada frase. Están básicamente centradas en estructuras sintácticas y limitadas a algunas operaciones de simplificación. Siddharthan propuso un sistema de simplificación que utiliza análisis de texto superficial generando un sistema muy eficiente. Max (2006) aplicó simplificación de textos en el proceso de escritura incorporando el sistema de simplificación en un procesador de textos.

También existen sistemas basados en aprendizaje automático que pueden aprender de un corpus operaciones de simplificación y también el grado de simplificación de la tarea (Petersen and Ostendorf, 2007).

Hay otros sistemas más generales que simplifican documentos como (Devlin and Tait, 1998), que utilizan técnicas de simplificación sustituyendo palabras no comunes por palabras más comunes. Otros sistemas, transforman frases en voz pasiva a voz activa (Canning, 2000), sistemas que realizan una reducción del número de claúsulas en las frases (Chandrasekar and Srinivas,

[2] http://www.plainlanguage.gov
[3] http://www.disabilityrightsfund.org/node/510
[4] http://www.w3.org/TR/WCAG20/

1997; Canning, 2000; Siddharthan, 2002) y finalmente sistemas que seleccionan adecuadamente las frases a nivel de discurso (Williams, Reiter, and Osman, 2003).

### 13.1.2 Simplificación de Textos Basado en Análisis de Dependencias

Aquí presentamos nuestro sistema propuesto que simplifica textos basando sus decisiones en un árbol de dependencias y un sistema simple basado en reglas, consiste en lo siguiente:

- Utilizar frases cortas.
- No intentar expresar más de una idea por frase.

Nuestro sistema usa como entrada un árbol de dependencias. El sistema, poda el árbol etiquetado centrándose las etiquetas de dependencias. Hemos evaluado el sistema con el corpus de español usado en la CoNLL-X Shared Task que está en el formato de datos presentado en la Sección 8.1.2, produciendo versiones simplificadas de las frases.

A continuación mostramos como con nuestros métodos es posible producir una versión simplificada de las frases mostrando ejemplos y diversas conclusiones.

#### Poda de los Árboles de Dependencias

En primer lugar y después de estudiar las etiquetas de dependencias que se encuentran en el corpus, nos preguntamos qué etiqueta/s de dependencias se podía/n eliminar para simplificar las frases, finalmente nos centramos en un conjunto de 3 etiquetas:

- "CC" (complemento circunstancial).
- "CD" (complemento directo).
- "CI" (complemento indirecto).

La razón es principalmente que este subconjunto aparece en la mayor parte de las frases, lo que posibilita hacer una simplificación agresiva. Después de varias comprobaciones, concluímos que la única etiqueta que se podía eliminar es la etiqueta "CC". Expresa información complementaria sobre una acción, como *cuándo, dónde, cómo*, y *por qué*. Pero esta etiqueta "CC" nunca aporta información sobre *quién* o *qué*. Por otro lado, eliminando las palabras que dependen de la mencionada etiqueta no siempre se pierde la información sobre *cuándo* o *cómo* ya que ese tipo de información no siempre depende de los verbos.

En la siguiente subsección mostramos nuestro algoritmo que elimina la etiqueta "CC", y los subárboles de las frases produciendo una versión simplificada de las frases.

### Algoritmo de Poda

Implementamos un algoritmo que requiere un árbol de dependencias en formato CoNLL y devuelve un texto plano con la versión simplificada de la frase de entrada.[5] Si el árbol de dependencias está bien formado, o al menos está correctamente anotado, la frase resultante será gramaticalmente correcta.

El algoritmo recorre los árboles de dependencias y hace lo siguiente:

1. El algoritmo elimina todos los nodos que tienen como etiqueta de dependencia la etiqueta "CC".
2. El algoritmo elimina todos los nodos que tenían como padre alguno/s de lo/s nodo/s eliminados en 1. El algoritmo itera en 2 hasta que ya no hay ningún nodo con algún padre eliminado.
3. Finalmente, genera un texto plano eliminando toda la información contenida en el corpus, quedándonos sólo con la etiqueta FORM.[6]

La Figura 13.1 muestra un ejemplo simple para la frase: *Tocó la vieja pared con cuidado.* La frase resultante es: *Tocó la vieja pared.* Nuestro algoritmo elimina la información sobre como él/ella tocó la pared, pero mantiene la información principal.

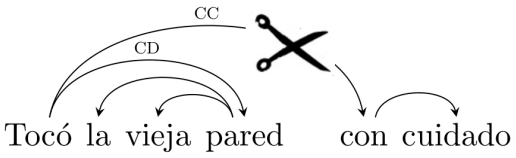



Figure 13.1: Poda de un árbol de dependencias con la frase: *Tocó la vieja pared con cuidado.*

Aquí mostramos otro ejemplo, la primera frase (1) es la versión original de una frase compleja incluída en el corpus. La segunda (2), es la salida del algoritmo para la frase (1):

1. *Tocó el familiar bulto con cuidado, recorriendo sus aristas con las yemas de los dedos, contemplando la imagen que le devolvía el espejo y pensando que todo aquello ya no tenía remedio, que nada podía hacer*

[5] Este algoritmo per se sólo puede funcionar con la anotación del corpus en español
[6] Mirar Sección 8.1.2

*ya por su cara, ni por su pecho, por esas piernas que no veía, pero sabía tan huesudas y separadas como las patas de un pollo mojado, y por esa carne blanquecina, fofa, que comenzaba a acumularse en torno a su cintura, a descolgarse hacia abajo arrastrando en su vértigo un ombligo progresivamente hondo, para añadir una nueva vejación, la de los años, a un cuerpo condenado de antemano, desde antes de existir, a ser feo.*

2. *Tocó el familiar bulto, recorriendo sus aristas, contemplando la imagen que le devolvía el espejo y pensando que todo aquello no tenía remedio, que nada podía hacer.*

Como podemos observar en el ejemplo, la versión simplificada es más fácil de leer y mantiene la información de la original.

**Estadísticas Globales sobre el Corpus**

En esta subsección mostramos estadísticas sobre el corpus con frases en español antes y después de aplicar nuestro algoritmo de simplificación. Por lo tanto, aplicamos el algoritmo frase a frase. El corpus contiene 3.512 frases, y el algoritmo simplificó 2.737 sentences (77,93%). Algunas frases no se simplificaron ya que muchas de las frases son cortas y sencillamente no tienen ningún árbol etiquetado con la etiqueta "CC". Los resultados del experimento se observan en la Tabla 13.1 que muestra el número de palabras, la longitud media por frase y la frase más larga de corpus originial y del corpus simplificado.

| | Original | Simplificado |
|---|---|---|
| # Palabras | 95.028 | 58.415 |
| Longitud Media Frase | 27,06 wf | 16,63 wf |
| Frase más larga | 143 wf | 94 wf |

Table 13.1: Estadísticas globales en el corpus, antes y después de la simplificación.

### 13.1.3 Evaluación

En esta Sección presentamos como evaluamos nuestro sistema de simplificación de textos

**Diseño de la Evaluación**

La evaluación está basada en dos encuestas. La primera consiste en personas adultas, con lo que no conforma el grupo objetivo de este tipo de sistemas.

La segunda consiste en un grupo de niños con 10 y 11 años, lo que conforma un grupo objetivo, ya que, es esperado que los niños puedan tener dificultad para comprender textos complejos.

En la primera encuesta, en la cual participaron 20 personas, preguntamos 4 cuestiones distintas sobre como de buena era la simplificación obtenida. Todos los participantes tenían estudios universitarios, y todos hablaban español como lengua materna. Ninguno de ellos sabía como funcionaba el sistema de simplificación. Les mostramos 20 frases con sus dos versiones, la frase original y la versión simplificada, les preguntamos lo siguiente, sólo podían responder, sí o no:

- Q1: *¿Se mantiene la idea principal de la frase?*
- Q2: *¿Toda la información eliminada era innecesaria?*
- Q3: *¿Sólo se han eliminado detalles sin importancia?*
- Q4: *¿Entiende mejor la versión simplificada o la versión original?*

Q2 y Q3 son bastante similares. Sin embargo, es bueno mencionar que en este tipo de encuentas es bueno repetr la misma pregunta formulada de manera distinta para asegurar la comprensión del encuestado.

Como segunda medida de evaluación, decidimos llevar a cabo una evaluación con 24 niños de 10 y 11 años. Seleccionamos 20 frases del corpus, y les mostramos la versión original y la simplificada. Como en la otra encuesta tenían que contestar sí o no a la siguiente pregunta: *¿Entiendes mejor la versión simplificada o la versión original?*

### Resultados y Discusión

La Tabla 13.2 muestra los resultados de la evaluación del grupo de adultos. En la Tabla, mostramos el porcentaje de respuestas 'sí' o 'no' para cada pregunta.

| Pregunta | Sí | No |
|---|---|---|
| Q1 | 67,58% | 32,42% |
| Q2 | 27,66% | 72,34% |
| Q3 | 46,72% | 53,28% |
| Q4 | 60,76% | 39,24% |

Table 13.2: Resultados obtenidos en la encuesta.

Consideramos que la primera pregunta (Q1) es la más importante. La encuesta nos devuelve un 67,58% de personas que dijeron sí. Podemos concluir que la mayoría de las personas pensó que en la mayoría de las frases, la idea principal se mantiene. Las frases donde la gente respondió "no" eran muy largas y nuestro sistema realizó simplificaciones muy agresivas y produjo que la mayor parte de la información de la frase se perdiese.

Si nos centramos en la segunda pregunta (Q2), la gente pensó que no toda la información eliminada era desechable. Sin embargo, los resultados de la pregunta Q1 nos indica, que a pesar de ello, la información más importante se mantiene, y parece inviable que en textos de menos de 100 palabras, exista algo totalmente eliminable, y además esa decisión tiene un carácter muy subjetivo.

Mirando los resultados de la tercera pregunta (Q3), podemos concluir que en algunas frases en las que perdemos información, no perdemos la más importante.

Si nos centramos en la cuarta preguna (Q4). Esta pregunta cuestiona como de bien se comprende la versión simplificada. La mayoría de los encuestados consideran que la versión simplificada se entiende mejor que la versión original. Además, es importante tener en cuenta que la mayoría de las frases no son realmente difíciles de entender en su versión original, y con ello podemos dar explicación al porcentaje de respuestas negativas a la pregunta.

Finalmente, como conclusión al experimento, vemos que la mayor parte de los encuestados consideran que la idea principal de las frases se preserva, lo que era nuestro primer objetivo, y además consideran que las versiones simplificadas son más faciles de leer y de entender, lo que era nuestro segundo objetivo.

Los resultados de la encuesta para los niños se presentan en la Tabla 13.3. Tuvimos 240 frases, lo que hace un total de 20 respuestas para cada frase. Los niños contestaron "sí" en 125 de los 240 casos. Lo que nos devuelve un porcentaje del 52,08%.

| Niños | Sí | No |
|---|---|---|
| 24 | 52,08% | 47,92% |

Table 13.3: Resultados obtenidos en la encuesta.

En esta segunda medida de evaluación los niños pueden tener el problema de entender las frases con fluidez, por ello nuestro sistema puede ayudarles a comprender las ideas principales de las mismas. El problema es que las frases seleccionadas podían incluir conceptos que hubiesen requerido una simplificación léxica más profunda, con lo cual, ni la frase simplificada sintácticamente ni la frase original eran fáciles de entender para ellos.

Podemos concluir que teniendo en cuenta ambas evaluaciones, nuestro

sistema ayuda a comprender las frases mejor, lo que en definitiva es el objetivo fundamental de este tipo de sistemas.

### 13.1.4 Conclusiones

La importancia del uso de simplificadores de textos para la educación o para gente con dificultades son evidentes. El impacto social de un sistema avanzado podría ser muy interesante.

El sistema que se presenta en esta Sección es una primera aproximación que atraviesa estructuras de dependencias podando las ramas menos relevantes, basándose, para ello, en la anotación sintáctica de las mismas. Con los resultados en la mano, concluímos que utilizando análisis de dependencias es posible simplificar las frases correctamente, al menos, en el caso particular del español. Sin embargo, con decisiones como las que presentamos, la simplificación puede ser muy agresiva produciendo pérdida de información relevante. Un refinamiento de las reglas de poda es la idea principal de trabajo futuro.

## 13.2 Inferir el Ámbito de La Negación para el Inglés Usando Análisis de Dependencias

La negación modifica por completo el significado de una frase, e indica la polaridad de las frases. Simple, en concepto, es un fenómeno complejo y ha sido un tema de investigación lingüística durante décadas (Horn, 1989). En la mayoría de los casos, la negación comprende una palabra que actúa como señal de negación y un sintagma negado conteniendo una o más palabras que están dentro del alcance de la negación. Hay además otro concepto que es el evento negado, que indica los conceptos negados dada una acción negativa, y también merece la pena su estudio.

A día de hoy, la detección de la negación es una tarea emergente en el procesamiento de lenguaje natural. Detectar, la negación es esencial en la mayoría de las tareas de minería de textos donde, en general, el objetivo es derivar conocimiento factual de los textos.

Un ejemplo donde la negación es un tema recurrente y relevante es en los documentos clínicos. La negación aparece con mucha frecuencia en este tipo de textos, y es por ello un fuente de errores a la hora de generar indexación automática de los mismos (Chapman et al., 2002). Por ello, y para mejorar la utilidad de la indexación automática de este tipo de documentos, parece necesario conocer si las palabras han sido negadas o no.

En esta Sección presentamos un sistema que anota el ámbito de la negación en frases escritas en inglés, haciendo uso de una técnica simple: se comprueba la presencia de alguna de las señales de negación que se encuen-

tran en un conjunto predefinido manualmente, y acto seguido un sistema de reglas anota las palabras que se encuentran dentro del alcance de esas señales de negación haciendo uso de estructuras de dependencias no etiquetadas. Hemos evaluado el sistema de dos maneras diferentes, en primer lugar utilizando Bioscope (Vincze et al., 2008), que es un corpus anotado con negaciones y además, participamos en la *SEM Shared Task donde pudimos evaluar con un corpus diferente (ver Sección 13.2.6)

### 13.2.1 Trabajo Relacionado

Existen dos tipos de sistemas que tratan la negación: sistemas que detectan palabras afectadas por negaciones, y (más recientemente) sistemas que clasifican el ámbito completo de la negación, lo que es una tarea más compleja. Nuestro sistema se enmarca en este segundo grupo. Existe además otro tipo de sistemas que tratan de extraer los eventos negados, como (Sarafraz and Nenadic, 2010), nosotros también, pero sólamente en nuestra participación en la *SEM Shared Task, ver la Sección 13.2.6.

En el dominio biomédico se ha investigado mucho la negación y su detección. Por ejemplo, Chapman et al. (2001) detectó negaciones e identificó términos médicos afectados por negaciones, utilizando un algoritmo simple basado en reglas, llamado NegEx. Consiguieron un 84,5% de precisión y un 77,8% de cobertura general evaluándolo con 400 frases seleccionadas aleatoriamente. De manera similar, Mutalik et al. (2001) reconoció patrones negados en los textos biomédicos utilizando un conjunto de 40 documentos médicos, el sistema fue denominado Negfinder y alcanzaron un 95,7% de cobertura y un 91,8% de precisión. Además, Huang y Lowe (2007) implementaron un enfoque híbrido para la detección de negaciones. Combinnaron expresiones regulares con análisis basado en gramáticas, alcanzando un 98,6% de precisión y un 92,6% de cobertura sobre un conjunto de test de 120 documentos.

Por otro lado, hay sistemas más complejos que infieren el ámbito de la negación. Esto es un problema más complejo ya que requiere determinar qué palabras están dentro del alcance de señales negativas, dónde abrir la señal de ámbito y dónde cerrarla. Uno de los sistemas principales, ha sido desarrollado por el equipo de Roser Morante (Morante, Liekens, and Daelemans, 2008; Morante and Daelemans, 2009), desarrollaro un sistema basado en aprendizaje automático. Evaluaron el sistema con el corpus Bioscope y sus resultados fueron de un 80,11% de precisión y un 78,44% de cobertura.

En 2010, se llevó a cabo un workshop sobre negación y especulación en el lenguaje natural (Morante and Sporleder, 2010) en Uppsala (Suecia). Uno de los objetivos principales fue encontrar los aspectos léxicos de la negación para definir como puede ser modelada para propositos computacionales. La mayoría de las aportaciones estaban orientadas al dominio biomédico. Cabe destacar el trabajo de Councill et al. (2010). Usaron también el corpus Bios-

cope para la evaluación, y su sistema estaba basado en conditional random fields con features basados en análisis de depedencias, alcanzando un 78,2% en cobertura y un 81,9% en precisión.

Más tarde, Zhu et al. (2010) presentaron un sistema basado en análisis semántico superficial (shallow parsing) y aprendizaje automático evaluado con el corpus Bioscope. Alcanzaron un 72,53% en cobertura y un 72,24% en precisión.

En La Tabla 13.4 resumimos los resultados de estos sistemas evaluados con el corpus Bioscope, ya que son comparables a los que mostramos en la primera parte de esta Sección.

| **Sistema** | **Evaluación** | **Cobertura** | **Precision** |
|---|---|---|---|
| (Morante and Daelemans, 2009) | Whole **Bioscope** | 78,44% | 80,11% |
| (Councill, McDonald, and Velikovich, 2010) | **Bioscope** Full Papers | 70,8% | 80,8% |
| (Councill, McDonald, and Velikovich, 2010) | Reviews | 78,2% | 81,9% |
| (Zhu et al., 2010) | Whole **Bioscope** | 72,53% | 72,24% |

Table 13.4: Resultados generales de sistemas que infieren el ámbito de la negación.

Finalmente, otro sistema interesante es el de Agarwal y Yu (2010). Su sistema está basado en conditional random fields e infieren el ámbito de señales de negación. Consiguieron un valor de la medida F1 del 98% y del 95% detectando señales de negación y su ámbito en el conjunto de documentos clínicos de Bioscope y un 97% y 85% con en el resto de colecciones.

### 13.2.2 Corpus Bioscope

Bioscope[7] (Szarvas et al., 2008) es un corpus de acceso libre, anotado manualmente con negaciones y su ámbito, con frases del dominio médico. Contiene más de 20.000 frases divididas en tres colecciones diferentes:

- Documentos clínicos. Consiste en 1954 documentos (6383 frases) que contienen informes clínicos.
- Artículos científicos. Consiste en 9 artículos científicos (2670 frases).
- Resúmenes de artículos científicos. Consiste en 1273 resúmenes (11871 frases) extraídos del corpus Genia (Kim et al., 2003).

La Tabla 13.5 muestra el número de documentos, frases con negación y el número de señales negativas de cada colección, así como la longitud media por frase y el porcentaje de scopes (ámbitos de negaciones) a la derecha y la izquierda en el corpus Bioscope.

[7] www.inf.u-szeged.hu/rgai/bioscope

| | Clinical | Papers | Abstracts |
|---|---|---|---|
| Documentos | 1,954 | 9 | 1,273 |
| Frases | 6,383 | 2,670 | 11,871 |
| Frases con Negación | 863 | 339 | 1597 |
| % Frases con Negación | 13.55 | 12.70 | 13.45 |
| Señales de Negación | 877 | 389 | 1848 |
| Longitud media por frase | 7.73 | 26.24 | 26.43 |
| % Ámbitos a la derecha | 97.64 | 81.77 | 85.70 |
| % Ámbitos a la izquierda | 2.35 | 18.22 | 14.29 |

Table 13.5: Estadísticas del corpus Bioscope centradas en la negación.

### 13.2.3 Encontrar el Ámbito de la Negación

Nuestro sistema consiste en dos algoritmos: el primero es capaz de inferir palabras afectadas por señales de negación (cues) atravesando árboles de dependencias generadas mediante el parser Minipar (Lin, 1998), y el segundo anota las frases con el ámbito de la negación utilizando la salida del primero. Este segundo algoritmo contiene una serie de reglas que se desarrollaron utilizando un conjunto de desarrollo, extraído de la colección de artículos científicos del corpus Bioscope, más concretamente con aproximadamente el 10% de los mismos.

Por lo tanto, nuestro sistema funciona de la siguiente manera: una frase analizada con Minipar y un léxico de señales negativas son la entrada del primer algoritmo (Affected Wordforms Detection Algorithm). Después el segundo algoritmo (Scope Finding Algorithm) utiliza un detector de voz pasiva y devuelve la frase anotada con el ámbito de la/s señal/es de negación.

#### Léxico de Señales de Negación

Consideramos los trabajos de Morante (2010) y Mutalik et al. (2001) para generar nuestro léxico estático, que se muestra en la Tabla 13.6. El léxico solo contiene la versión canónica (lema) de las palabras, pero el sistema es capaz de generar formas derivadas.

| | | | |
|---|---|---|---|
| not | no | neither..nor | none |
| discard | rule out | fail | avoid |
| absence | lack (v) | lack (n) | without |
| unable | rather than | absent | cannot |

Table 13.6: Léxico de señales negativas.

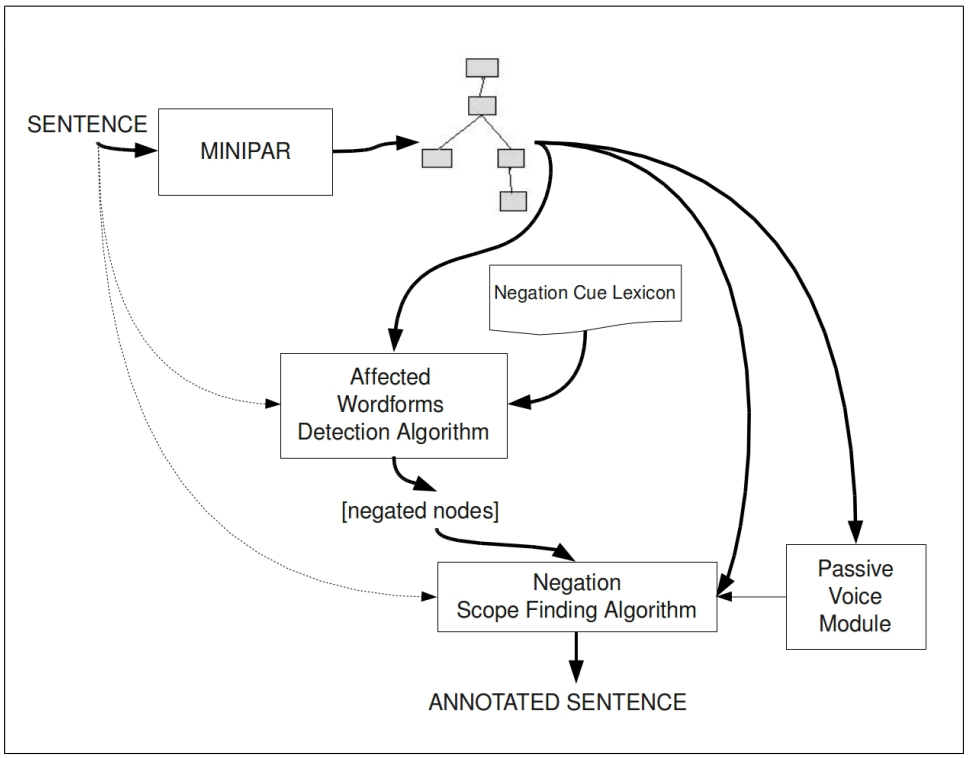



Figure 13.2: Arquitectura del sistema.

### Algoritmo que Detecta Palabras Afectadas por Negaciones

Implementamos un algoritmo que usa un árbol de dependencias para una frase dada y devuelve para cada señal de negación el conjunto de palabras afectadas por la misma. Utiliza para ello el conjunto de señales de negación que se muestra anteriormente.

El algoritmo atraviesa un árbol de dependencias para una frase, y lleva a cabo los siguientes pasos:

1. Detecta en la frase, si alguno de los nodos están contenidos en el léxico de señales de negación.

   - Si la señal de negación es un verbo, lo marca como señal de negación.
   - Si la señal de negación no es un verbo, el algoritmo marca el verbo (si existe) afectado por la señal. De ese modo, se pueden detectar las palabras que no dependen directamente de la señal de negación.

2. Para el resto de nodos, si alguno depende directamente de algunas de las palabras marcadas anteriormente, el sistema lo marca como negado.

La negación se propaga hasta encontrar los nodos terminales, con lo que palabras que no están directamente relacionadas con las señales también se pueden detectar.

El algoritmo finalmente genera un conjunto de nodos que contienen las palabras que están afectadas por las señales de negación para cada frase.

### Algorimo que Anota el Ámbito de la Negación

El segundo algoritmo se implementó utilizando un conjunto de desarrollo que es un subconjunto de la colección de artículos del corpus Bioscope. Concretamente, el primer 10% de las frases.

Este algotitmo utiliza el conjunto de palabras devueltas por el Affected Wordforms Detection Algorithm (primer algoritmo) y el árbol de dependencias para anotar las frases y decidir donde debe abrirse el ámbito de una señal de negación y dónde cerrarlo.

Conviene tener en cuenta que en el corpus Bioscope, cuando el ámbito se abre a la derecha de la señal de negación, se deja el sujeto de la frase fuera. Esto corresponde a frases en voz activa y son muy frecuentes en el corpus. Pero por otro lado, las frases en voz pasiva y para algunas señales como la dupla *neither ... nor*, abren el ámbito a la izquierda de la señal.

Por lo tanto, lo primero que se debe hacer es decidir donde se abre el ámbito de la señal de negación y para ello desarrollamos un sistema capaz de detectar la voz pasiva en las frases, lo segundo que se debe hacer es decidir donde cerrarlo, pero eso en este caso es sencillo, porque en nuestro caso, consiste en mapear la salida del algoritmo anterior.

Nuestro sistema decide si una frase (negativa) está en voz pasiva, si y sólo si:

- Contiene un verbo transitivo, como *show, consider, see, use, detect*, etc.
- Sigue alguno de los siguientes patrones:
  1. *modal verb + <u>not</u> + be + past participle.*
  2. *am/is/are/was/were + <u>not</u> + past participle.*
  3. *have/has been + <u>not</u> + past participle.*

Una vez que nuestro sistema ha decidido si la frase está en voz pasiva o no, el algoritmo itera uno a uno los nodos de la frase y aplica el siguiente conjunto de reglas. Las reglas se aplican en el orden presentado abajo, y solamente se puede aplicar una regla por nodo.

1. Apertura del ámbito:

   **a.** Si el nodo está contenido en el conjunto de nodos marcados como negados por el Affected Wordforms Detection Algorithm y el ámbito para la señal correspondiente no está abierto: el sistema entonces abre la señal de ámbito y establece que el ámbito para esa señal de negación ya está abierto.

   **b.** Si el nodo es una señal de negación y la frase está en voz pasiva: el sistema va hacia atrás y abre el ámbito justo antes del sujeto de la frase, encontrando una nombre, primer determinante o el último símbolo de puntuación.

   **c.** Si el nodo es una señal de negación y la frase no está en voz pasiva: el sistema abre el ámbito justo antes del nodo.

2. Cierre del ámbito:

   **a.** Si el nodo es una señal de puntuación, seguida por algunas palabras que indican el inicio de otra frase o idea, como *but*: el sistema cierra el ámbito justo después del nodo.

   **b.** Si el nodo es cualquier palabra y todos los nodos que están marcados como negados para la señal de negación ya están incluidas en el ámbito: el sistema cierra el ámbito justo antes del nodo.

   **c.** Si el nodo es la señal de fin de frase: el sistema cierra el ámbito justo después del nodo.

3. Si ninguna de estas reglas se aplica, el nodo se añade a la frase directamente.

En este punto, el sistema ha computado el ámbito (o ámbitos) de la negación (o negaciones) para una frase. La Figura 13.3 ilustra el proceso para la siguiente frase: *The reason why the two other families were not detected is more complex.*

### 13.2.4 Evaluación

Aquí presentamos la evaluación llevada a cabo y el diseño de la misma.

#### Diseño de la Evaluación

Seleccionamos Bioscope como corpus de evaluación porque en el momento del desarrollo de este sistema no había más corpora anotados con negaciones y sus ámbitos. En la Sección 13.2.6 mostramos nuestra participación en una Shared Task, lo que nos proporcionó la oportunidad de evaluarlo con un corpus distinto.

El primer paso para la evaluación consistió en seleccionar las frases contenidas en Bioscope, considerando que para la colección de artículos científicos debíamos eliminar las frases utilizadas para el desarrollo, ver

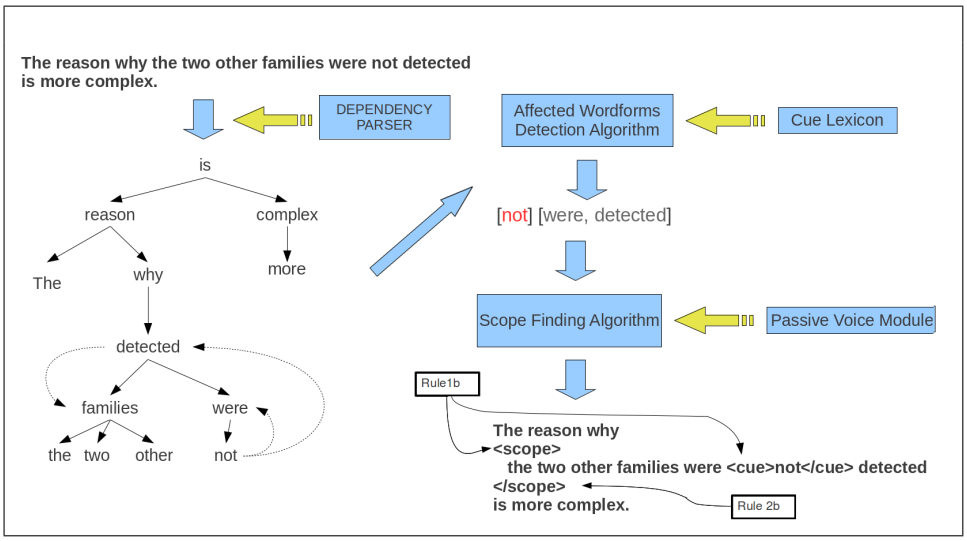



Figure 13.3: El proceso que realiza nuestro sistema para una frase ejemplo.

Sección 13.2.3. Por lo tanto, para evaluar utilizamos el 100% de los documentos clínicos, el 90% de los artículos y el 100% de los resúmenes.

Además, utilizamos las siguientes medidas de evaluación:

$$P(Precision) = \frac{Tokens \quad correctly \quad negated \quad by \quad our \quad system}{Tokens \quad negated \quad by \quad the \quad system}$$

$$R(Recall) = \frac{Tokens \quad correctly \quad negated \quad by \quad our \quad system}{Tokens \quad negated \quad in \quad the \quad collection}$$

$$F1 = \frac{2PR}{P+R}$$

$$PCS = \frac{Correct \quad Scopes \quad annotated \quad by \quad our \quad system}{Scopes \quad annotated \quad in \quad the \quad collection}$$

$$PCNC = \frac{Correct \quad negation \quad Cues \quad annotated \quad by \quad our \quad system}{Negation \quad Cues \quad annotated \quad in \quad the \quad collection}$$

### Resultados y Discusión

Cuando analizamos las tres colecciones de Bioscope nuestro sistema produce los resultados dados en la Tabla 13.7. Es bueno tener en cuenta que la entrada de nuestro sistema es texto completamente plano, sin ninguna información anotada.

| Collection | Precision | Recall | F1 | PCS | PCNC |
|---|---|---|---|---|---|
| Papers | 73.49% | 80.70% | 76.93% | 56.43% | 91.15% |
| Abstracts | 84.92% | 84.03% | 84.48% | 68.92% | 95.56% |
| Clinical | 95.83% | 90.58% | 93.13% | 89.06% | 94.82% |

Table 13.7: Resultados de nuestro sistema.

La principal razón por la que la colección de documentos clínicos produce mejores resultados es básicamente que las frases contenidas en esa colección son realmente mucho más simples que el resto, teniendo una longitud media de 7,73 palabras, mientras que las otras dos colecciones tienen frases largas con más de 26 palabras (Morante and Daelemans, 2009). Además en las dos colecciones derivadas de artículos científicos las estructuras sintácticas son más difíciles, teniendo muchas de ellas, más de una señal de negación.

Por una razón similar, los resultados de la colección de resúmenes son mejores ya que las frases contenidas en este tipo de textos suelen ser más simples que las que se encuentran en el cuerpo de los artículos científicos.

### Comparación con otros Sistemas

A continuación mostramos una comparación aproximada con algunos sistemas del estado del arte. Lo comparamos con el sistema de Morante y Daelemans (2009), el sistema basado en parsing semántico superficial de Zhu et al. (2010) y el sistema basado en dependencias de Councill et al. (2010). La comparativa se muestra en la Tabla 13.8.

| Collection | System | Precision | Recall | F1 | PCS | PCNC |
|---|---|---|---|---|---|---|
| Papers | Our Results | 73.49% | **80.70%** | **76.93%** | **56.43%** | 91.15% |
| | Morante et Al. | 72.21% | 69.72% | 70.94% | 41.00% | **92.15%** |
| | Zhu et Al. | 56.27% | 58.20% | 57.22% | – | – |
| | Councill et Al. | **80.80%** | 70.80% | 75.50% | 53.70% | – |
| Abstracts | Our Results | **84.92%** | **84.03%** | **84.48%** | **68.92%** | **95.56%** |
| | Morante et Al. | 81.76% | 83.45% | 82.60% | 66.07% | 95.09% |
| | Zhu et Al. | 78.24% | 78.77% | 78.50% | – | – |
| Clinical | Our Results | **95.83%** | **90.58%** | **93.13%** | **89.06%** | 94.82% |
| | Morante et Al. | 86.38% | 82.14% | 84.20% | 70.75% | **97.72%** |
| | Zhu et Al. | 82.22% | 80.62% | 81.41% | – | – |

Table 13.8: Resultados de la comparativa entre los distintos sistemas, evaluados con las tres colecciones de Bioscope.

El sistema de Morante está basado en aprendizaje automático, sin embargo, nuestro sistema está basado en reglas utilizando un conjunto de desarrollo. Con lo cual, los conjuntos de test no son exactamente los mismos, el sistema de Morante se evaluó utilzando 10-fold cross validation para

abstracts y utilizaron la colección para el entrenamiento del sistema en la evaluación de las otras dos colecciones. Por ello, consideramos que sus resultados están directamente marcados por este hecho, haciendo descender los resultados para tanto documentos clínicos como para artículos científicos. Sin embargo, en esta comparativa, tratamos de hacer los resultados tan comparables como nos fue posible. Por una razón similar, nuestros resultados para artículos científicos tienen más opción de ser mejores.

De manera parecida, el sistema de Councill et al. sólo llevaron a cabo el experimento con artículos científicos y además en su entrenamiento había frases de un sistema derivado de minería de opiniones, con lo cual, teniendo en cuenta este hecho, sus resultados son realmente elevados.

En la Tabla 13.9 mostramos el porcentaje de ámbitos correctos (PCS) por cada señal de negación, para señales de negación que ocurren 10 ó más veces en cada colección. Comparamos nuestros resultados con los del sistema presentado por Morante and Daelemans (2009). Merece la pena destacar, que para tanto nuestro sistema como el de ellos, las señales de negación con peor PCS tienen un elevado porcentaje de ámbitos a la izquierda (*absent, unable*). En este caso, consideramos todo el conjunto de test incluyendo los datos del conjunto de desarrollo, con lo que nuestros resultados para artículos científicos deben ser observados bajo esa óptica. De ese modo podíamos comparar de manera directa con el sistema de Morante et al.

| | Abstracts | | | Papers | | | Clinical | | |
|---|---|---|---|---|---|---|---|---|---|
| | # | **Mor.** | **Our** | # | **Mor.** | **Our** | # | **Mor.** | **Our** |
| **absence** | 57 | 56.14 | **71.93** | – | – | – | – | – | – |
| **absent** | 13 | 15.38 | **38.46** | – | – | – | – | – | – |
| **cannot** | 28 | **42.85** | 28.57 | 16 | 50.00 | 50.00 | – | – | – |
| **fail** | 57 | 63.15 | **85.97** | 13 | 38.46 | **53.84** | – | – | – |
| **lack** | 85 | **57.64** | 52.94 | 20 | 45.00 | **50.00** | – | – | – |
| **neither** | 33 | 51.51 | **72.72** | – | – | – | – | – | – |
| **no** | 207 | 73.42 | **81.64** | 44 | 50.00 | **54.54** | 673 | 73.10 | **89.60** |
| **none** | – | – | – | 10 | 0.00 | **71.42** | – | – | – |
| **not** | 1036 | **69.40** | 66.41 | 200 | 39.50 | **64.50** | 57 | 50.87 | **66.66** |
| **rather than** | 20 | 65.00 | 65.00 | 12 | **41.66** | 25.00 | – | – | – |
| **unable** | 30 | 40.00 | **73.33** | – | – | – | – | – | – |
| **without** | 82 | **89.02** | 79.27 | 24 | 58.33 | **70.83** | – | – | – |

Table 13.9: PCS por señal de negación, para aquellas que ocurren 10 ó más veces. La columna # indica el número de apariciones para cada caso; la columna **Our** muestra los resultados de nuestro sistema; y la columna **Mor.** los resultados del sistema de Morante et al.

En la tabla se observa como el sistema de Morante et al. no es capaz de cubrir señales de negación con una elevada frecuencia en el conjunto de test y una baja frecuencia en el conjunto de entrenamiento, como es el caso de

*none* en la dupla abstracts-papers. Sin embargo, nuestro sistema trata cada señal como un evento separado y es por ello, por lo que es capaz de anotar mejor en estos casos.

Finalmente, cabe destacar que no hemos incluido en la comparativa el sistema de Agarwal y Yu (2010), cuyos resultados describimos en la Sección 13.2.1. En su evaluación, con resultados muy buenos, utilizaron una forma de evaluación distinta, incluyendo las frases sin señales de negación como aciertos. Esto se discutió en el tutorial presentado en la conferencia IJCNLP en 2011 en Chiang Mai, Tailandia[8], sin embargo, lo consideramos un trabajo a tener en cuenta por sus resultados y la facilidad de acceso que proporciona.

### 13.2.5 Conclusiones

En esta Sección hemos presentado un sistema capaz de inferir el alcance de las senáles de negación. Por estos resultados, podemos concluir que el análisis de dependencias puede ser una herramienta válida para ello, al menos, en el caso particular del inglés.

Finalmente y por otro lado, consideramos que el ámbito de la negación no debe ser siempre continuo, como es el caso de la anotación que se encuentra en Bioscope. Este tipo de anotación debería incluir la posibilidad de añadir palabras que están en claúsulas diferentes, como se observa en la Figura 13.4.

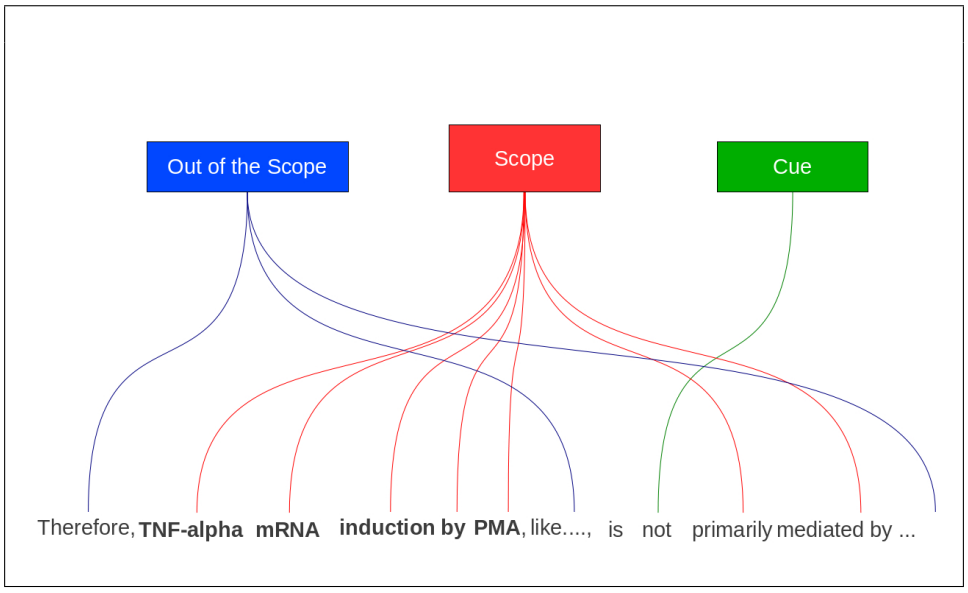



Figure 13.4: Enfatizando la idea de anotar el ámbito de manera que sea posible anotar ámbitos discontinuos.

[8] http://www.ijcnlp2011.org/ijcnlp2011/downloads/tutorial/tu3_present.pdf

### 13.2.6 La Participación en el *SEM Shared Task Challenge

La *SEM Shared Task[9] (Morante and Blanco, 2012) estaba basada en inferir y clasificar el ámbito y los eventos asociados a negaciones, para ello se proporcionó un corpus de entrenamiento y un corpus de desarrollo basado en historias de Conan Doyle (Morante and Daelemans, 2012). En la Shared Task, hubo dos tipos de competiciones:

1. La competición abierta, donde los desarrolladores podían usar tanto sus propios recursos como la anotación proporcionada en el corpus.
2. La competición cerrada, donde los desarrolladores debían usar la anotación del corpus.

Presentamos el sistema de las secciones anteriores a la Shared Task, modificando la salida siendo capaces de anotar las frases del mismo modo que en el corpus proporcionado en la Shared Task.

#### El Corpus de Conan Doyle

El corpus usado en la Shared Task consiste en algunos capítulos de obras de Conan Doyle (Morante and Daelemans, 2012), con distintos niveles de anotación. Se anotó de manera similar a los corpora de la CoNLL Shared Task[10] usando un formato tabular para los ámbitos y el evento negado. La Figura 13.5 muestra un ejemplo. Las columnas contienen la siguiente información:

1. Número de capítulo e identificador.
2. Contador de frase.
3. Contador de token.
4. Palabra.
5. Versión canónica (lema) de la palabra.
6. Categoría gramatical.
7. Análisis sintáctico basado en constituyentes.
8. Señal de negación. La palabra si se trata de una señal, en otro caso, un valor nulo.
9. Ámbito. La palabra si pertenece a algún ámbito, en otro caso, un valor nulo.

[9] http://www.clips.ua.ac.be/sem2012-st-neg/
[10] Ver sección 8.1.2 y la Figura 8.1.

10. Evento negado. La palabra si es un evento negado, en otro caso, un valor nulo.

Si la frase tiene más de un ámbito, el primero se anota en las columnas 8, 9 y 10, el segundo sería en las columnas 11, 12 y 13 y así sucesivamente. Las siguientes columnas contienen la misma información que las columnas 8, 9, 10 (mostradas en la lista de arriba) pero con información relativa a otras señales de negación de la misma frase. Este tipo de anotación permite ámbitos discontinuos y anidados, y esto es algo que no era posible con la anotación plana de Bioscope.

```
baskervilles01  8       0       Holmes  Holmes  NNP     (S(S(NP*)       _       _       _
baskervilles01  8       1       was     be      VBD     (VP*    _       _       _
baskervilles01  8       2       sitting sit     VBG     (VP*    _       _       _
baskervilles01  8       3       with    with    IN      (PP*    _       _       _
baskervilles01  8       4       his     his     PRP$    (NP*    _       _       _
baskervilles01  8       5       back    back    NN      *))     _       _       _
baskervilles01  8       6       to      to      TO      (PP*    _       _       _
baskervilles01  8       7       me      me      PRP     (NP*)))))       _       _       _
baskervilles01  8       8       ,       ,       ,       *       _       _       _
baskervilles01  8       9       and     and     CC      *       _       _       _
baskervilles01  8       10      I       I       PRP     (S(NP*) _       I       _
baskervilles01  8       11      had     have    VBD     (VP*    _       had     _
baskervilles01  8       12      given   give    VBN     (VP*    _       given   given
baskervilles01  8       13      him     him     PRP     (NP*)   _       him     _
baskervilles01  8       14      no      no      DT      (NP(NP* no      _       _
baskervilles01  8       15      sign    sign    NN      *)      _       sign    _
baskervilles01  8       16      of      of      IN      (PP*    _       of      _
baskervilles01  8       17      my      my      PRP$    (NP*    _       my      _
baskervilles01  8       18      occupation      occupation      NN      *)))))) _       occupation
baskervilles01  8       19      .       .       .       *)      _       _       _
```

Figure 13.5: La frase "Holmes was sitting with his back to me, and I had given him no sign of my occupation" anotada en el corpus de Conan Doyle.

Los participantes de la Shared Task tenían al principio dos corpora distintos: un conjunto de desarrollo y un conjunto de entrenamiento, haciendo un total de 3899 frases teniendo 1056 de ellas negaciones. Una vez que los participantes envíamos el sistema, dispusimos además de un tercer conjunto *ciego* de test para la evaluación.

### Nuestro Sistema

A la Shared Task presentamos una modificación del sistema presentado en la Sección 13.2. El sistema está basado en reglas, con lo que tuvimos que modificar algunas de ellas manualmente para hacer posible el tratamiento de estructuras de negación incluidas en el corpus de Conan Doyle y los nuevos retos que representa, como un conjunto más completo de señales. Esta sección por tanto ejemplifica los problemas de los sistemas basados en reglas a la hora de modificarlos cuando cambia la tarea y/o el dominio.

Nuestro sistema presentado a la Shared Task está basado en las siguientes propiedades:

- Utiliza un léxico estático de señales de negación.
- Utiliza un algoritmo que atraviesa señales de negación (el Affected Wordforms Detection Algorithm, presentado en la Sección 13.2.3).
- Clasifica el ámbito de las negaciones usando un sistema de reglas que es el que se presentó en la Sección 13.2.3 pero con una serie de modificaciones.
- Aplica unas reglas muy simples para la clasificación del evento negado. El evento negado, que no se trataba en el sistema previo, indica los conceptos negados de la frases. Por ejemplo, en la frase, *No mention of that local hunt*, la palabra *mention* es el evento negado relacionado con la señal *No*.
- No utiliza la información sintáctica proporcionada en el corpus de Conan Doyle (solamente para los eventos negados). Utilizamos, por lo tanto, nuestro analizador de dependencias. Es por ello, por lo que participamos en la competición abierta.

Nuestro (nuevo) sistema consiste en cinco modulos diferenciados:

- Un léxico estático.
- Un algoritmo que dado un árbol de dependencias y un léxico de señales de negación produce un conjunto de palabras afectadas por las negaciones.
- Un sistema basado en reglas que produce la anotación del ámbito de la frase estudiada.
- Un sistema de post-procesamiento que hace uso de la salida del sistema inicial y produce la salida esperada.
- Un sistema simple basado en reglas para la anotación del evento negado.

El léxico que contiene las señales de negación es estático, tal y como el que se usó en el sistema previo. Sin embargo, este nuevo léxico es mucho mayor, con la idea de ser capaces de cubrir todas las señales de negación que aparecían en el conjunto de desarrollo, tuvimos que generar un léxico con 152 palabras, que se muestra por completo en el Apéndice E, y la Tabla 13.10 contiene una pequeña parte del mismo. El léxico presentado anteriormente contenía menos de 20.

El algoritmo que procesa los árboles de dependencias (Affected Wordforms Detection Algorithm) es el mismo que usamos en la versión anterior. Sin embargo, el algoritmo de anotación de ámbitos (The Scope Finding Algorithm) fue modificado para poder producir la anotación requerida, funciona de la siguiente manera:

| | | |
|---|---|---|
| not | no | neither..nor |
| unnecessary | unoccupied | unpleasant |
| unpractical | unsafe | unseen |
| unshaven | windless | without |

Table 13.10: Parte del léxico.

- El sistema abre un ámbito cuando encuentra una nueva señal de negación que fué detectada por el Affected Wordforms Detection Algorithm. En el corpus Conan Doyle, la mayor parte de las frases contienen el sujeto en el ámbito. Con lo cual, ampliamos la regla que restringía ese comportamiento sólo a las frases en voz pasiva (para Bioscope) a todas las frases, con algunas excepciones que se describen abajo.

  Con lo cual, para la mayor parte de las señales de negación el sistema busca el sujeto y lo anota dentro del ámbito. Como hemos mencionado, hay algunas excepciones a esta regla, y son básicamente por la presencia de algunas señales como *without* o *neither...nor*. Para estas señales, el sistema abre el ámbito en la señal.

- El sistema cierra el ámbito cuando no hay más palabras para añadir, i.e:

  - Encuentra alguna palabra que indica otra cláusula, como *but* o *because*.
  - No hay más palabras a consumir de la salida del primer algoritmo.
  - Fin de frase.

- Además añadimos una nueva regla que permite el tratamiento de señales de negación que son prefijo o sufijo de otra palabra, como *meaning-less*: si el sistema encuentra una palabra como esa, entonces anota el sufijo o el prefijo como señal de negación (en el ejemplo, sería *less*) y el resto de la palabra como parte del ámbito. El Affected Wordforms Detection algorithm detecta toda la palabra como señal de negación.

Con la idea de producir una solución que pudiera proporcionar algunos resultados en el tratamiento de los eventos negados, desarrollamos el siguiente sistema de reglas:

- Cuando la señal de negación contiene un prefijo o sufijo negativo, anotamos la palabra como evento negado.

- Cuando la señal de negación es o *not* o *n't*[11] y la siguiente palabra es un verbo, de acuerdo a la anotación de la categoría gramatical del corpus Conan Doyle, el verbo se anota como evento negado.

Para poder producir la salida con el formato de la Shared Task, hicimos un sistema de post procesamiento que procesa la frase anotada con el estilo de Bioscope (mostramos un ejemplo como aclaración: *<scope>There is <cue>no</cue> problem</scope>*) para poder producir la salida esperada. Iteramos la frase, palabra a palabra y hacemos lo siguiente:

- Si la palabra contiene la cadena *<scope>*, el sistema inicia un nuevo ámbito reservando tres nuevas columnas y colocando la palabra en la primera columna de ámbito libre.
- Si la palabra contiene la cadena *<cue>*, el sistema la coloca en la siguiente columna para señales de negación libre.
- Si la palabra está anotada como evento negado, el sistema la coloca en la siguiente columna para eventos negados libre.

Es importante destacar que estas tres reglas pueden dispararse para el mismo elemento, es decir, no son exclusivas.

### Resultados y Discusión

Aquí mostramos los resultados en dos tablas distintas: Tabla 13.11 muestra los resultados del sistema con el conjunto de test, Tabla 13.12 muestra los resultados del sistema con el conjunto de desarrollo. Estas tablas muestran las diferentes métricas de evaluación que se describen abajo:

- `tp`. True positive, lo que significa que el sistema predijo la señal, el evento o el ámbito correctamente.
- `fp`. False positive, lo que significa que el sistema predijo la señal, el evento o el scope, pero no aparece como tal en el corpus de evaluación.
- `fn`. False negative, lo que significa que el sistema no predijo una señal, un evento o un scope, que sin embargo aparece en el corpus de evaluación.
- Precision, Recall (cobertura) y F1 también los describimos en la Sección 13.2.4, pero en este caso es referente a ámbitos, no a palabras. Salvo para la medida Scope tokens que es referente a palabras.
- Las columnas *gold* y *system* contienen el número de apariciones en el gold standard (*gold*) y el número de predicciones (*system*).

| **Test set** | gold | system | tp | fp | fn | precision (%) | recall (%) | F1 (%) |
|---|---|---|---|---|---|---|---|---|
| Cues: | 264 | 235 | 170 | 39 | 94 | 81.34 | 64.39 | 71.88 |
| Scopes (cue match): | 249 | 233 | 96 | 47 | 153 | 67.13 | 38.55 | 48.98 |
| Scopes (no cue match): | 249 | 233 | 96 | 48 | 152 | 66.90 | 38.96 | 49.24 |
| Scope tokens (no cue match): | 1805 | 2096 | 1222 | 874 | 583 | 58.30 | 67.70 | 62.65 |
| Negated (no cue match): | 173 | 81 | 36 | 42 | 134 | 46.15 | 21.18 | 29.03 |
| Full negation: | 264 | 235 | 29 | 39 | 235 | 42.65 | 10.98 | 17.46 |

Table 13.11: Resultados en el corpus de test.

| **Development** | gold | system | tp | fp | fn | precision (%) | recall (%) | F1 (%) |
|---|---|---|---|---|---|---|---|---|
| Cues: | 173 | 161 | 115 | 16 | 58 | 87.79 | 66.47 | 75.66 |
| Scopes (cue match): | 168 | 160 | 70 | 17 | 98 | 80.46 | 41.67 | 54.90 |
| Scopes (no cue match): | 168 | 160 | 70 | 17 | 98 | 80.46 | 41.67 | 54.90 |
| Scope tokens (no cue match): | 1348 | 1423 | 1012 | 411 | 336 | 71.12 | 75.07 | 73.04 |
| Negated (no cue match): | 122 | 71 | 35 | 31 | 82 | 53.03 | 29.91 | 38.25 |
| Full negation: | 173 | 161 | 24 | 16 | 149 | 60.00 | 13.87 | 22.53 |

Table 13.12: Resultados en el corpus de desarrollo.

Como podemos observar, los resultados en el conjunto de desarrollo son mejores que los que obtenemos para el conjunto de test. La razón es simple, utilizamos el conjunto de desarrollo para la generación de las reglas con lo que el comportamiento del sistema es más similar a lo que se encuentra en el mismo.

Nuestro sistema sólo es capaz de detectar algunas de las señales de negación (entre de 72% F1 y 76% F1). Con lo cual, uno de los mayores problemas es que el léxico de negaciones es estático. Esto no era un problema grave en el sistema basado en Bioscope, porque el conjunto de señales contenido en el corpus era más limitado.

Es bueno destacar también, que para la medida *Scope tokens*, que mide token a token, en lugar de scope a scope (ámbito a ámbito) nuestro sistema produce interesantes resultados (alrededor de 63% F1 y 73% F1, respectivamente). Esto evidencia que nuestro sistema proviene de otro dominio y su comportamiento es bastante positivo, pero tiene problemas con decisiones generales de cierre y apertura de ámbitos.

Se puede observar que los resultados para los *negated events* (o eventos negados) son muy bajos (alrededor del 20% F1), lo cual es esperado, ya que nuestro sistema de reglas para este caso es muy limitado y simple.

El sistema se clasificó cuarto en un grupo de cinco sistemas, en la *Open Track*. La Tabla 13.13 muestra los resultados de los cinco sistemas, que son:

- Uio2 (Lapponi et al., 2012), basado en aprendizaje automático, utilizando análsis de dependencias sintáctico y semántico.

[11] Las contracciones están incluidas en el corpus de Conan Doyle, pero en Bioscope no

- UGroningen (r1 y r2) (Basile et al., 2012), tiene dos sistemas distintos, ambos basados en estructuras de discurso.
- UCM-1 (Carrillo de Albornoz et al., 2012), un sistema basado en reglas y parsing de constituyentes.
- UCM-2 (Ballesteros et al., 2012), nuestro sistema.

| System | Cues | | | Scopes (cue match) | | | Scopes (no cue match) | | | Scope Tokens | | | Negated | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Prec. | Rec. | F1. | Prec. | Rec. | F1. | Prec. | Rec. | F1. | Prec. | Rec. | F1. | Prec. | Rec. | F1. |
| UiO2 | 89.17 | 93.56 | **91.31** | 85.71 | 62.65 | **72.39** | 85.71 | 62.65 | **72.39** | 82.25 | 82.16 | **82.20** | 66.90 | 57.90 | **61.79** |
| UGroningen r2 | 88.89 | 84.85 | 86.82 | 76.12 | 40.96 | 53.26 | 76.12 | 40.96 | 53.26 | 69.20 | 82.27 | 75.17 | 56.63 | 65.29 | 60.65 |
| UCM-1 | 89.26 | 91.29 | 90.26 | 82.86 | 46.59 | 59.64 | 82.86 | 46.59 | 59.64 | 85.37 | 68.53 | 76.03 | 66.67 | 12.72 | 21.36 |
| **UCM-2 (ours)** | 81.34 | 64.39 | 71.88 | 67.13 | 38.55 | 48.98 | 66.90 | 38.96 | 49.24 | 58.30 | 67.70 | 62.65 | 46.15 | 21.18 | 29.03 |
| UGroningen r1 | 86.90 | 82.95 | 84.88 | 46.38 | 12.85 | 20.12 | 46.38 | 12.85 | 20.12 | 69.69 | 70.39 | 69.99 | 53.94 | 52.05 | 52.98 |

Table 13.13: Resultados de los sistemas presentados a la Shared Task.

Comparando estos resultados, se evidencia que nuestro sistema sufre por el léxico estático de señales de negación. Sin embargo, se observa que, a pesar de esa diferencia, no produce resultados muy inferiores al resto, estando más o menos en el rango del resto de sistemas. Exceptuando el primero de todos. Sería interesante probar un reconocedor de señales diferente y comprobar como de lejos podemos llegar con nuestro anotador de ámbitos.

El resto de sistemas fueron desarrollados desde cero para la Shared Task, sin embargo, nuestro sistema es una modificación de un sistema previo. Y en el caso de un sistema basado en reglas, donde la intención inicial es mantener la estructura del sistema al máximo, se convierte en una tarea complicada.

Por otro lado, aparte de errores derivados del léxico, la mayoría de errores se producen por dificultades en la anotación de frases con más de un ámbito. El sistema anota este tipo de frases al estilo de Bioscope, pero en el corpus de Conan Doyle esto es mucho más complicado, ya que los ámbitos pueden ser discontinuos.

### Conclusiones sobre la Participación a la Shared Task

Como conclusión principal cabe decir que modificar este tipo de sistemas para actuar con un tipo diferente de textos y diferentes niveles de anotación es complicado. Sin embargo, teniendo esto en cuenta y los resultados obtenidos, podemos decir que nuestro sistema es competitivo.

El sistema podría mejorarse de diferentes maneras:

- Mejorar el tratamiento de frases con más de un ámbito.
- Reemplazar el analizador de dependencias por otro, que produzca resultados a la altura del estado del arte.
- Proponer diferentes formas de conseguir un léxico no estático, capaz de producir mejores resultados.

## 13.3 Resumen del Capítulo

En este Capítulo, hemos mostrado una serie de trabajos de aplicación donde hacemos uso de árboles de dependencias. Hemos estudiado la simplificación de textos y la inferencia del ámbito de señales de negación, proporcionando dos sistemas basados en reglas que están principalmente basados en procesar estructuras sintácticas de dependencias.

# Chapter 14

# Conclusiones y Trabajo Futuro

Este último capítulo resume las conclusiones principales y sugerencias de trabajo futura de esta tesis. Hemos estudiado análisis de dependencias desde distintas perspectivas:

1. Estudio del problema centrándonos en el tamaño de los corpora y la longitud de las frases.
2. Mejora de los analizadores.
3. Optimización de los analizadores, o los también llamados, modelos de aprendizaje automático.
4. Aplicación de estructuras de dependencias para resolver problemas de PLN.

Nuestra idea inicial era cerrar el círculo generando contribuciones en distintos aspectos del análisis de dependencias, y creemos que hemos conseguido nuestros objetivos.

Como decimos en la Sección 8.2 el objetivo de esta tesis era contestar a las siguientes preguntas:

- ¿Qué factores del corpus de entrenamiento pueden afectar la precisión en la fase de análisis?
- ¿Es posible mejorar la precisión manipulando el corpus de entrenamiento?
- ¿Es posible mejorar la precisión combinando diferentes analizadores cuyo origen es el mismo generador de analizadores?
- ¿Es posible modificar el comportamiento de los modelos basados en transiciones con la intención de mejorar su comportamiento?

- ¿Es posible modificar el comportamiento de los modelos basados en grafos con la intención de mejorar su comportamiento?
- ¿Es posible automatizar la optimización de los analizadores de dependencias?
- ¿Es posible crear un sistema preciso y automático de selección de features para un sistema basado en transiciones? o incluso, ¿un sistema basado en aprendizaje automático?
- ¿Podemos utilizar estructuras de dependencias para resolver problemas del procesamiento del lenguaje natural?

En las siguientes secciones, damos respuesta a cada una de ellas.

## 14.1 Conclusiones Sobre el Estudio del Análisis de Dependencias

En esta Sección damos nuestras respuestas, basadas en nuestras conclusiones, sobre los estudios y resultados obtenidos que se presentan en el Capítulo 10 titulado *Estudios Iniciales sobre Análisis de Dependencias*:

- ¿Qué factores del corpus de entrenamiento pueden afectar la precisión en la fase de análisis?

Hay varios factores en los corpora de entrenamiento que son muy importantes para la fase de análisis y la fase de entrenamiento, como por ejemplo, la longitud de las frases y el tamaño de los corpora. Sin embargo, en la Sección 10.1 y especialmente en la Sección 10.3 demostramos que no siempre tener más palabras conlleva tener mejores resultados, obteniendo como conclusión que la riqueza de las frases, e incluso más importante el nivel de anotación, afecta de manera significativa la precisión final. Además, hemos demostrado que las frases largas (en Sección 10.2) son muy ricas para el entrenamiento de un analizador basado en transiciones, porque incluyen diversas estructuras sintácticas que llevan al analizador a aprender más transiciones. Hemos mostrado evidencias de que las frases largas afectan la precisión final porque los analizadores tienen problemas analizándolas, como vimos en la Sección 10.4. Y además, hemos sugerido que para el desarrollo de futuros corpora se debe tener en cuenta las frases que se van a incluir, ya que parece que las frases que comparten la estructura sintáctica con otras no aportan nada a la precisión final.

Finalmente, en la Sección 10.4, hemos evaluado un conjunto importante de analizadores de maneras distintas, centrándonos en las medidas de ajuste completo por frase (o complete-match scores), evidenciando que la modificación del punto de vista muestra problemas que no se destacan con las otras

medidas de evaluación, como la importancia de la longitud de las frases y problemas de algunos analizadores.

- ¿Es posible mejorar la precisión manipulando el corpus de entrenamiento?

Sí, podemos afirmar que la manipulación del corpus de entrenamiento puede modificar la precisión de manera significativa. En la Sección 10.1 vimos que la precisión es homogénea, sin embargo, parece inteligente seleccionar las frases que van a ser incluidas con cuidado, porque podría haber diferencias significativas. Un corpus rico de frases largas es, por lo tanto, rico en diversas estructuras sintácticas y por lo tanto podemos esperar mejores resultados.

## 14.2 Conclusiones sobre la Mejora del Análisis de Dependencias

En esta Sección damos nuestras respuestas sobre las preguntas mostradas abajo, teniendo en cuenta los resultados y estudios del Capítulo 11 titulado *Mejora del Análisis de Dependencias*:

- ¿Es posible mejorar la precisión combinando diferentes analizadores cuyo origen es el mismo generador de analizadores?

Esta pregunta está relacionada con el estudio de viabilidad mostrado en la Sección 11.1, donde estudiamos como entrenar algunos analizadores específicos con el objetivo de analizar algunas palabras donde los analizadores fallan frecuentemente, hemos demostrado que la idea es, de hecho, viable. Sin embargo, la propuesta falla en un factor principal, porque confiamos en como están anotadas las frases en el corpus de entrenamiento. A pesar de ello, el estudio muestra que es interesante tratar de mejorar la precisión centrándonos en una pequeña parte de las frases generando resultados significativos ya que las palabras seleccionadas (preposiciones, nexo y conjunciones) son muy frecuentes y por lo tanto muy importantes ya que actuan como conectores en las frases y por lo tanto en las estructuras de dependencias.

- ¿Es posible modificar el comportamiento de los modelos basados en transiciones con la intención de mejorar su comportamiento?

- ¿Es posible modificar el comportamiento de los modelos basados en grafos con la intención de mejorar su comportamiento?

Estas dos preguntas tienen que ver con el trabajo donde estudiamos la posición de la raíz de los árboles durante el análisis y el entrenamiento,

mostrado en la Sección 11.2. Creemos que hemos desarrollado interesantes conclusiones mostrando que la posición de la raíz es muy relevante, especialmente importante para algunos algoritmos basados en transiciones, ampliamente usados donde los arcos hacia la izquierda se generan de manera voraz, como Nivre arc-eager.[1] Creemos que nuestros hallazgos muestran que la posición de la raíz debe ser tenida en cuenta en los desarrollos de futuros algoritmos de análisis. Sin embargo, también es verdad, que cambiar la posición de la raíz, o forzar al parser a trabajar sin raíz, en algunos algoritmos basados en transiciones (donde los arcos a la izquierda no se generan de manera voraz) o en algoritmos basados en grafos (como los modelos de MSTParser) no produce una mejora significativa, generando sólo ruido entre un escenario y otro. En el estudio nos centramos en el algoritmo Nivre arc-eager y el algoritmo Nivre arc-standard porque ambos conforman los dos órdenes de análisis como se puede ver en las Figuras 12.2 y 12.3 y la discusión sobre ellos durante el Capítulo 12.

Como conclusión principal podemos afirmar que hay mucha capacidad de mejora. Lo hemos demostrado durante la tesis, mostrando estudios sobre dos problemas diferentes plenamente relacionados con el análisis de dependencias.

## 14.3 Conclusiones Sobre Optimización del Análisis de Dependencias

En esta Sección damos respuesta a la pregunta que se muestra abajo y que está relacionada con lo que se estudia durante el Capítulo 12 titulado *Optimizando el Análisis de Dependencias*:

- ¿Es posible automatizar la optimización de los analizadores de dependencias?

Viendo el resultado de MaltOptimizer, que es el sistema que se presenta durante el Capítulo 12, una respuesta directa a la pregunta que se muestra arriba es, definitivamente, sí. Hemos demostrado que es posible automatizar el proceso de optimizar MaltParser centrándonos en los parámetros preliminares, algoritmos de análisis y más importante hemos visto que la selección automática de features es viable y produce buenos resultados. Hemos visto que es incluso posible obtener mejores resultados que aquellos generados mediante optimización manual, incluso optimización manual llevada a cabo por investigadores muy experimentados en la materia, ya que además el espacio de búsqueda es muy grande para llevarse a cabo de manera manual.

MaltOptimizer es un sistema experto, lo que nos ha llevado a aprender como funciona un analizador basado en transiciones, y las cosas distintas

[1] Mirar, la Sección 9.1.1.

que representan. Realmente, esperamos que nuestro sistema va a tener un gran interés por la comunidad dada la importancia de MaltParser aparte de los resultados que hemos conseguido con ello.

También hemos demostrado que con este sistema es posible optimizar sólo un algoritmo (mirar Apéndice D), parando el proceso entre fases y ejecutando la búsqueda de features para el algoritmo deseado, consiguiendo buenos resultados para cada una de las ventanas donde se añaden y se quitan features de los modelos de features por defecto.

- ¿Es posible crear un sistema preciso y automático de selección de features para un sistema basado en transiciones? o incluso, ¿un sistema basado en aprendizaje automático?

MaltOptimizer dispone de un algoritmo de selección de features óptimos dado un conjunto de entrenamiento y un algoritmo de parsing. Por lo tanto, la respuesta a esta pregunta es, otra vez, sí. Hemos demostrado que una búsqueda automática de features es posible y hemos demostrado que es posible conseguir resultados muy interesantes.

## 14.4 Conclusiones sobre la Aplicación del Análisis de Dependencias

En esta Sección damos respuesta a la pregunta mostrada abajo teniendo en cuenta los resultados del Capítulo 13 titulado *Aplicando el Análisis de Dependencias*:

- ¿Podemos utilizar estructuras de dependencias para resolver problemas del procesamiento del lenguaje natural?

Teniendo en cuenta los resultados de los experimentos y los sistemas presentados en el Capítulo 13, la respuesta es definitivamente sí. Hemos demostrado que las estructuras de dependencias pueden ser muy útiles para resolver problemas relacionados con el procesamiento del lenguaje natural.

En la Sección 13.1 demostramos que las estructuras de dependencias son realmente útiles para simplificar frases que han sido previamente analizadas con un analizador de dependencias asegurando la correción gramatical de la salida. Además, en la Sección 13.2, mostramos que es posible explorar los árboles para encontrar las palabras que están bajo el alcance de señales de negación. Los resultados de ambos sistemas muestran que los sistemas basados en reglas que utilizan análisis de dependencias, como herramienta, son totalmente viables.

Sin embargo, es bueno tener en cuenta que estos sistemas basados en reglas tienen un problema importante, ya que es difícil su extensión a nuevos dominios o diferentes formatos de etiquetado como se ve en la Sección 13.2

cuando modificamos nuestro sistema basado en Bioscope para tratar frases que provienen de un dominio distinto y participar así en la *SEM Shared Task.

Como conclusión final, podemos enfatizar que el análisis de dependencias es muy útil y puede ser utilizado como una herramienta para resolver distintos problemas similares a los que presentamos en esta tesis.

## 14.5 Trabajo Futuro

En el futuro próximo, nos gustaría seguir trabajando en estos temas pero de una forma diferente. Nos gustaría aplicar lo que hemos aprendido durante la elaboración de esta tesis. Uno de los problemas que nos motiva en mayor medida es la aplicación de estructuras sintácticas a textos obtenidos de redes sociales. Consideramos que una buena idea podría ser la aplicación de nuevos analizadores que sean capaces de anotar información sintáctica y semántica y aplicar, de una manera novedosa, algunas de las ideas que hemos desarrollado durante la tesis como la extracción de la información negada en los textos. Además creemos, que la experiencia adquirida en la selección de features, optimización y aprendizaje automático puede aportar una interesante forma de llevar a cabo estos experimentos.

Merece la pena comentar que podría ser interesante investigar y/o implementar analizadores robustos y universales sin necesidad de anotar diferentes corpora para diferentes lenguajes, en lugar de eso, tener un analizador universal capaz de proporcionar resultados competitivos para todos los idiomas. El primer paso debería ser una anotación universal sobre la anotación de frases en los corpora. Ya existe trabajo en este sentido, iniciado por Petrov, Das y McDonald (2012).

A continuación mostramos trabajo específico para cada una de estas perspectivas que estudiamos en la presente tesis:

### 14.5.1 Estudio del Análisis de Dependencias

Sería interesante llevar a cabo experimentos similares como los que se exponen en el Capítulo 10, con corpora diferentes y analizadores distintos. Sin embargo, es verdad que esta parte de la tesis nos sirvió para comprender el comportamiento de los analizadores. Pero, sin duda, este tipo de experimentos deben siempre llevarse a cabo en las etapas iniciales con la idea de establecer los conocimientos necesarios.

### 14.5.2 Mejora del Análisis de Dependencias

Sin duda alguna, todavía hay muchísimo margen de mejora en los analizadores de dependencias. Los mejores analizadores obtienen, a día de hoy, resultados que rondan los 80% - 90% LAS, por lo tanto todavía hay mucho

camino hasta generar modelos que fueran capaces de no producir errores. Realmente, creemos que se podrán generar nuevos algoritmos y modelos capaces de producir mejores resultados, sobre todo, teniendo en cuenta que la capacidad de computación irá aumentando.

Considerando nuestro trabajo que combina analizadores, pueden existir nuevas maneras de encontrar combinaciones de las salidas (o entradas) de los analizadores. Sin embargo, también es verdad que hay mucho trabajo hecho sobre esto, como vimos en la Sección 9.3, con lo que parece poco probable que nosotros (u otros investigadores) puedan producir sistemas híbridos más precisos, si no se mejoran previamente los sistemas base.

Otra idea que estaría muy bien desarrollar en el futuro es el estudio y modificación de los algoritmos actuales. Siguiendo, por ejemplo, ideas similares a las aplicadas en la Sección 11.2. Por otro lado, el tiempo de ejecución de algunos analizadores quizá podría reducirse, produciendo modelos capaces de generar resultados en mucho menos tiempo y de ese modo, evitar las complicaciones que tienen muchos de los sistemas.

### 14.5.3 Optimización del Análisis de Dependencias

Nos gustaría explorar algoritmos de selección automática de features de manera similar a lo que ya hemos empezado en el Apéndice C. Esto podría requerir el desarrollo de estrategias de optimización más avanzadas que sean capaces de lidiar con la correlación que existen entre los algoritmos, features y/o parámetros.

Aparte de eso, consideramos que la rama de optimización podría ser extendida y aplicada a diferentes analizadores de dependencias que utilicen lenguajes similares de especificación de features. En el caso de los sistemas basados en transiciones, podría ser bastante automático generar una traducción de nuestros modelos a los de otro tipo de sistema. Para analizadores basados en grafos, los features no se definen a través de las estructuras de datos, si no por palabras, lo que lo hace más simple. Con lo cual, parece posible, generar un sistema automático de optimización de features para otros sistemas.

Finalmente, nuestra experiencia adquirida en selección de features podría ser interesante en un algoritmo que fuese capaz de funcionar online, incorporando nuestro sistema de selección de features que fuera capaz de actualizar de manera automática los valores requeridos.

### 14.5.4 Aplicación del Análisis de Dependencias

Considerando nuestro sistema de simplificación de textos, podemos definitivamente ampliar buena parte del mismo, refinando las reglas sintácticas, o incorporando reglas de carácter léxico o semántico. El sistema que se presenta en esta tesis es solamente un primer paso que nos sirve como ejemplo

para mostrar la utilidad de los analizadores de dependencias.

Y considerando el sistema que es capaz de inferir el ámbito de las negaciones , podría ser de nuevo posible mejorarlo utilizando un sistema de aprendizaje automático o analizadores de dependencias del estado del arte, como es el caso de MaltParser o MSTParser. Otras ideas podrían ser las siguientes:

- Utilizar uno de los sistemas de la CoNLL 2009 Shared Task (Hajič et al., 2009), como (Bohnet, 2009), utilizando de ese modo la información semántica que proporcionan.
- Utilizar un detector de cláusulas como (Carreras, Màrquez, and Castro, 2005) o algunos de los sistemas de la CoNLL 2001 Shared Task (Tjong Kim Sang and Déjean, 2001).

Pensamos que estas tecnologías proporcionan información que es muy útil para inferir el ámbito de las señales de negación.

## 14.6 Resumen del Capítulo

En este último Capítulo del manuscrito hemos mostrado las principales conclusiones y sugerencias de trabajo futuro, teniendo en cuenta nuestra experiencia reciente. Hemos resumido todas las preguntas introducidas en el inicio de este manuscrito y hemos buscado respuesta a las mismas utilizando los resultados de cada uno de los estudios, experimentos y trabajos presentes en la tesis.

Finalmente, hemos mostrado algunas sugerencias de trabajo futuro estudiando lo que podremos hacer en los próximos años, mostrando interesantes ideas en cada una de las ramas de la tesis, donde nuestra experiencia podría ayudarnos a a llevarlos a cabo.

# Appendix A

# CoNLL-X Shared Task Results

In this appendix we show the results of the systems presented to the CoNLL-X Shared Task computing LAS.[1] The results are shown in Table A.1. The list shown above may serve as a guide to find the results in the table. We show in bold the results of MaltParser ((7) (Nivre et al., 2006b)), and the Spanish treebank results because they are more relevant to this thesis.

- (1) Canisius et al. (Canisius et al., 2006).
- (2) Giuseppe Attardi (Attardi, 2006).
- (3) YuChieh Wu (Wu, Lee, and Yang, 2006).
- (4) Carreras et al. (Carreras, Surdeanu, and Màrquez, 2006).
- (5) Deniz Yuret (Yuret, 2006).
- (6) Eckhard Bick (Bick, 2006).
- (7) Nivre et al. (Nivre et al., 2006b).
- (8) Michael Schiehlen (Schiehlen and Spranger, 2006).
- (9) Jinshan Ma et al. (Liu et al., 2006).
- (10) Dreyer et al. (Dreyer, Smith, and Smith, 2006).
- (11) Chang et al. (Chang, Do, and Roth, 2006).
- (12) Johansson et al.(Johansson and Nugues, 2006).
- (13) McDonald et al. (McDonald, Lerman, and Pereira, 2006).

[1] Please, see Section 3.4 in which we show a reclassification of the same parsers by using different evaluation measures

- (14) Riedel et al. (Riedel, Çakici, and Meza-Ruiz, 2006).
- (15) Kenji Sagae et al. (Sagae et al., 2007).
- (16) Nobuyuki Shimizu (Shimizu, 2006).
- (17) Simon Corston-Oliver (Corston-Oliver and Aue, 2006).
- (18) Yuchang Cheng (Cheng, Asahara, and Matsumoto, 2006).

| Ara | Chi | Cze | Dan | Dut | Ger | Jap | Por | Slo | **Spa** | Swe | Tur | AV | SD | Bul | Name |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 57.64 | 78.37 | 60.92 | 77.90 | 74.59 | 77.56 | 87.41 | 77.42 | 59.19 | **68.32** | 79.15 | 51.07 | 70.80 | 11.11 | 78.74 | (1) |
| 53.81 | 54.89 | 59.76 | 66.35 | 58.24 | 69.77 | 65.38 | 75.36 | 57.19 | **67.44** | 68.77 | 37.80 | 61.23 | 9.92 | 72.89 | (2) |
| 63.81 | 74.81 | 59.36 | 78.38 | 68.45 | 76.52 | 90.11 | 81.47 | 67.83 | **72.99** | 71.72 | 55.09 | 71.71 | 9.67 | 79.73 | (3) |
| 60.94 | 83.68 | 68.82 | 79.74 | 67.25 | 82.41 | 88.13 | 83.37 | 68.43 | **77.16** | 78.65 | 58.06 | 74.72 | 9.72 | 83.30 | (4) |
| 52.42 | 72.72 | 51.86 | 71.56 | 62.75 | 63.82 | 84.35 | 70.35 | 55.06 | **69.63** | 65.23 | 60.31 | 65.01 | 9.46 | 73.49 | (5) |
| 55.37 | 76.18 | 63.02 | 74.61 | 69.51 | 74.74 | 84.75 | 78.18 | 64.31 | **71.37** | 74.09 | 53.87 | 70.00 | 9.25 | 79.21 | (6) |
| **66.71** | **86.92** | **78.42** | **84.77** | **78.59** | **85.82** | **91.65** | **87.60** | **70.30** | **81.29** | **84.58** | **65.68** | **80.19** | **8.53** | **87.41** | **(7)** |
| 44.39 | 66.20 | 53.34 | 76.05 | 72.11 | 68.73 | 83.35 | 71.01 | 50.72 | **46.96** | 71.10 | 49.81 | 62.81 | 13.01 | 0.00 | (8) |
| 50.74 | 75.29 | 58.52 | 77.70 | 59.36 | 68.11 | 70.84 | 71.13 | 57.21 | **65.08** | 63.83 | 41.72 | 63.29 | 10.42 | 67.64 | (9) |
| 53.37 | 71.63 | 60.54 | 66.61 | 61.56 | 70.97 | 82.87 | 75.28 | 58.73 | **67.62** | 67.58 | 46.05 | 65.23 | 9.93 | 74.81 | (10) |
| 60.92 | 85.05 | 72.88 | 80.60 | 72.91 | 84.17 | 89.07 | 83.99 | 69.52 | **79.72** | 82.31 | 60.51 | 76.80 | 9.43 | 0.00 | (11) |
| 64.29 | 72.49 | 71.46 | 81.54 | 72.67 | 80.43 | 85.63 | 84.57 | 66.43 | **78.16** | 78.13 | 63.39 | 74.93 | 7.65 | 0.00 | (12) |
| 66.91 | 85.90 | 80.18 | 84.79 | 79.19 | 87.34 | 90.71 | 86.82 | 73.44 | **82.25** | 82.55 | 63.19 | 80.27 | 8.43 | 87.57 | (13) |
| 66.65 | 89.96 | 67.44 | 83.63 | 78.59 | 86.24 | 90.51 | 84.43 | 71.20 | **77.38** | 80.66 | 58.61 | 77.94 | 10.05 | 0.00 | (14) |
| 62.71 | 84.73 | 75.24 | 81.56 | 76.61 | 84.92 | 90.37 | 86.01 | 69.06 | **77.68** | 82.00 | 63.21 | 77.84 | 8.95 | 0.00 | (15) |
| 62.83 | 0.00 | 0.00 | 75.81 | 0.00 | 0.00 | 0.00 | 0.00 | 64.57 | 73.17 | **79.49** | 54.23 | 34.18 | 36.26 | 0.00 | (16) |
| 63.53 | 79.92 | 74.48 | 81.74 | 71.43 | 83.47 | 89.95 | 84.59 | 72.42 | **80.36** | 79.69 | 61.74 | 76.94 | 8.47 | 83.36 | (17) |
| 65.19 | 84.27 | 76.24 | 81.72 | 71.77 | 84.11 | 89.91 | 85.07 | 71.42 | **80.46** | 81.08 | 61.22 | 77.70 | 8.67 | 86.34 | (18) |
| 59.94 | 78.32 | 67.17 | 78.31 | 70.73 | 78.58 | 85.86 | 80.63 | 65.16 | **73.52** | 76.44 | 55.95 | | | 79.98 | Average |

Table A.1: Results of the systems in the CoNLL-X Shared Task, we show in bold the results of MaltParser and the Spanish results.

# Appendix B

# Replicating the Experiment Shown in Section 3.3 with Complete-Match Scores

In the experiment shown in Section 3.3 we could observe that the lexical level is not as critical as the syntactic level for accurate training due not only to the little number of lexical features considered for training but also to the absence or presence of some phenomena, such as declination in some languages. The words contained in the training corpora are important for inflected languages or morphologically rich languages that use case to encode grammatical relations, so wordforms contain information that must be considered, as referred in (Herrera and Gervás, 2008). Given that the errors produced by the models shown in the experiment explained in Section 3.3 are not the same for every model, we believe that complete–match results should be analyzed too.

Therefore, the results of the experiment introduced in Section 3.3 for LCM are shown in Table B.1. In Figure B.1 we show the behavior shown by the models for LCM.[1]

As we can observe in Figure B.1, the highest LCM is achieved very soon when a significant amount of sentences is already included in the iterative training corpus. And to include more wordforms inside the training corpora does not contribute at all to the final parsing accuracy. We got the same conclusion in Section 3.3.

As we said in Section 3.4 it is also important to take into account that the languages with a shorter average sentence length in the testing data set are the ones with a higher LCM after parsing. We can observe in this appendix that the hypothesis introduced in Section 3.3 can be also demonstrated when considering complete-match measures. When we reach a significant quantity

[1] In Section 3.4 we show a complete experiment with sentence-based measures and several parsers of the CoNLL-X Shared Task.

| Language | 10% | 20% | 30% | 40% | 50% | 55% | 60% | 65% | 70% | 75% | 80% | 85% | 90% | 95% | 100% |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Arabic | 0.00 | 1.37 | **1.37** | 2.05 | 2.74 | **1.37** | **2.74** | **2.05** | **2.05** | **1.37** | **1.37** | 9.59 | **9.59** | **9.59** | **9.59** |
| Bulgarian | 1.01 | 13.32 | 24.12 | 26.38 | 26.63 | 28.90 | **28.90** | 29.15 | 30.15 | 31.16 | 33.42 | **33.17** | 33.92 | **32.66** | **32.66** |
| Chinese | 29.98 | 51.56 | 55.94 | 59.86 | 62.97 | 64.13 | 64.59 | **64.13** | 64.70 | 65.28 | 66.09 | 67.13 | 67.70 | 67.94 | 68.51 |
| Czech | 0.27 | 18.08 | 19.45 | 20.27 | 22.46 | **22.46** | **22.46** | 24.11 | 24.93 | **24.93** | **24.93** | **23.83** | **24.93** | **24.93** | 25.02 |
| Danish | 0.62 | 15.84 | 19.88 | 22.36 | **22.36** | **22.04** | 23.60 | **22.98** | 23.29 | 24.53 | 25.16 | **24.22** | **24.84** | **23.60** | **24.53** |
| Dutch | 12.18 | 19.43 | 19.69 | 19.95 | **19.17** | 21.24 | 21.76 | **21.76** | 22.02 | 22.54 | 23.06 | **23.06** | **23.06** | **23.06** | 25.38 |
| German | 3.36 | 24.37 | 29.13 | 30.25 | **29.97** | 31.93 | 33.61 | 33.89 | 35.29 | **33.61** | **33.61** | **33.61** | **35.01** | **34.35** | **35.29** |
| Japanese | 36.53 | 68.97 | 70.52 | 71.80 | **71.80** | 72.36 | 72.92 | 73.91 | **73.91** | **73.62** | 74.47 | **74.19** | 74.90 | 75.18 | 75.60 |
| Portuguese | 1.74 | 14.28 | **14.28** | 15.68 | 16.37 | **16.37** | **16.37** | 17.42 | **17.42** | **17.42** | 18.82 | 19.16 | **19.16** | **19.16** | **19.16** |
| Slovene | 0.00 | 4.74 | **4.74** | 6.98 | 7.98 | **7.98** | 10.22 | 10.47 | 10.72 | **10.47** | **10.72** | **9.73** | **9.73** | **10.72** | **10.47** |
| Spanish | 0.00 | 7.77 | 12.14 | 14.56 | **14.56** | 15.53 | **15.53** | **13.59** | **12.62** | 14.56 | 16.50 | 16.90 | 18.93 | **16.50** | **17.96** |
| Swedish | 6.43 | 21.85 | 23.91 | 25.19 | 25.44 | 25.71 | 28.02 | 28.53 | 29.56 | 30.33 | **29.56** | 31.62 | **31.36** | **31.62** | **31.36** |
| Turkish | 0.16 | 6.58 | 7.54 | 7.70 | 9.31 | 9.47 | **8.98** | 9.79 | 9.95 | 10.59 | **10.43** | **10.43** | **9.95** | **9.79** | **10.11** |
| Average | 7.69 | 22.35 | 25.23 | 26.92 | 27.65 | 28.29 | 29.14 | 29.32 | 29.72 | 30.03 | 30.68 | 31.39 | 31.92 | **31.59** | 32.14 |

Table B.1: Labeled Complete Match (LCM) obtained by the iterative models trained with the reduced amount of wordforms corpora. We show in bold the cases in which the result is lower than (or the same) as a previous iteration.

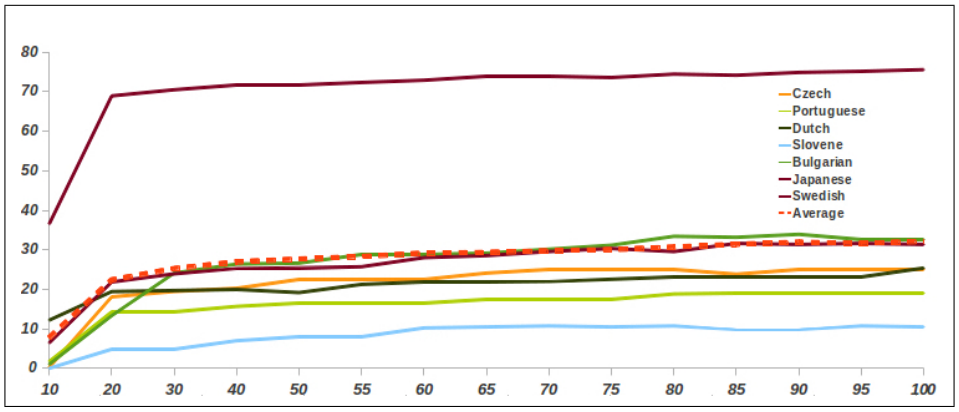



Figure B.1: Stable behavior that all the training corpora show when an iterated training experiment is carried out considering Labeled Complete Match (LCM).

of wordforms, the results may be even worse than training with the whole corpus. For instance, considering the Spanish treebank, we may observe that using the 90% of the training corpus we obtained 18.93% LCM, while the 95% training model obtained 16.50% LCM, which is more than 2% worse, and using 100% of the training corpus we obtained 17.96% LCM, which is 1% worse. Smaller corpora may be therefore even more accurate than bigger ones in some cases. The Spanish case is only an example, remind that all the data shown in bold in Tables 3.5 and B.1, show the cases in which a model trained with smaller corpora obtained better accuracy (or the same) than the current one for LAS and LCM, respectively.

# Appendix C

# Yet Another Feature Selection Experiment

In this Appendix we describe some experiments that emphasize how we decided to produce the definite feature selection experiments shown in Section 5.2.3. We show two different algorithms that follow the six steps presented in Section 5.2.3:

- **The Relaxed Greedy** approach traverses all the 6 steps, shown in Section 5.2.3, adding one feature at a time and keeping the feature set that provides the best result so far. This Relaxed Greedy algorithm tries with all the backward and forward operations for all the steps. Therefore, it can be understood as an exhaustive feature search that adds two, three or even more features at a time. The number of experiments of this algorithm is quite high because it just adds and tries with a big set of experiments, keeping the best feature model after each attempt, we could therefore expect that this algorithm overfits the performance in some cases providing a model with higher training error but lower test error.
- **The Greedy Algorithm**. It tries to minimize the number of experiments according to a heuristic derived from proven experience (Nivre and Hall, 2010) and following the six steps shown at the beginning of Section 5.2.3. Therefore, in spite of trying with all the big set of possible features for each step as it is done in the Relaxed Greedy algorithm, it does the following:

    1. If backward selection provides improvements for a specific window, we do not try forward selection for this window.
    2. As soon as forward selection is unsuccessful for a specific window, we do not try further forward selection experiments for this window.

This algorithm could intuitively get stuck in a local optima because it reduces the number of experiments, we could therefore expect that it might underfit the performance. However, we will see in this Appendix that it is not the case. In the following Section we show an in-depth comparison between the Greedy and the Relaxed Greedy algorithms taking the results into account, we also show which algorithm is the most accurate in order to get an optimal feature set.

In Section C.1, we show the experiments that we carried out with the algorithms introduced above, showing the outcomes of each one and explaining the results of the experiments. In Section C.2, we also show which way is the best to produce a K-fold cross validation for this kind of experiments, which is basically the one included in MaltOptimizer and explained during Chapter 5.

## C.1 Do the Algorithms Overfit or Underfit the Performance?

As we mentioned above, the Greedy and the Relaxed Greedy algorithm could underfit and overfit the performance respectively, we took these two facts as hypotheses. In order to see whether these hypotheses have been falsified or corroborated we are going to compare both algorithms by using the training set, and also trying with the original division of training and test corpora that was used during the CoNLL-X Shared Task in which we test with an unseen test set not used during the optimization.

Therefore, In this section we firstly show the results of each of the algorithms implemented during the training phase in which we get the optimal feature set, afterwards, we show the real case. Finally, and considering the outcomes of the first two experiments, we perform a 5-fold cross validation strategy to demonstrate its usefulness.[1] During this experiments we also compare our results with the results given by the default models to see whether our methods are useful or not.

### C.1.1 First Results and Comparisons between Greedy and Relaxed Greedy

Table C.1 shows the results of the Greedy algorithm and the Relaxed Greedy algorithm for a selection of languages from the CoNLL-X Shared Task. Note that these results are obtained using 80% of the training set for training and 20% as development test set, which were obtained using the entire training set and a separate held-out test set for evaluation.

[1] See, Section C.2 in which we show in which cases this is useful from our experience.

| Language | DefaultFM | Greedy | Relaxed Greedy |
|---|---|---|---|
| Arabic | 63.84 | 65.56 (+1.72) | **66.00 (+2.16)** |
| Dutch | 78.02 | **82.63 (+4.61)** | 82.49 (+4.47) |
| Slovene | 68.40 | 71.71 (+3.31) | **72.43 (+4.03)** |
| Spanish | 76.64 | 79.38 (+2.74) | **79.62 (+2.98)** |
| Swedish | 83.50 | 84.09 (+0.59) | **84.20 (+0.70)** |
| Turkish | 58.29 | 66.92 (+8.63) | **67.19 (+8.90)** |

Table C.1: Labeled attachment score with comparison to default feature model and the greedy approach for a selection of languages from the CoNLL-X shared task.

We can observe how the Relaxed Greedy approach beats the results of the Greedy one, with the exception of Dutch. Nevertheless, the differences are not very remarkable, because the Relaxed Greedy algorithm always carries out more than 100 different experiments, and the Greedy algorithm between 40 and 50, depending on the results of each one.

This fact means, that the decisions that we took during the development of the Greedy algorithm seem to be the correct ones. This fact is also evidenced in the Figures C.1 and C.2 in which we show the results of the Greedy and the Relaxed Greedy algorithms for the Turkish and Slovene corpus.[2] We can see how the Greedy algorithm achieves an optimal accuracy much more faster than the Relaxed Greedy one, but in some cases it could get stuck in local optima because the Relaxed Greedy approach beats these results finding eventually a more complex and accurate feature configuration.

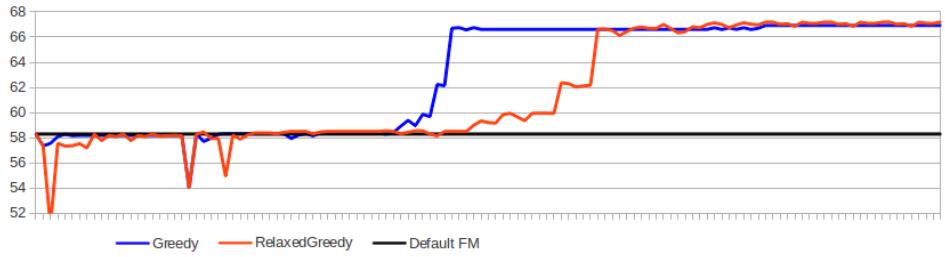



Figure C.1: Results obtained by Greedy and Relaxed Greedy in every step of the algorithms using the Turkish treebank.

[2]In the Figures we simulate that the Greedy algorithm is waiting for the Relaxed Greedy before it starts a new step.

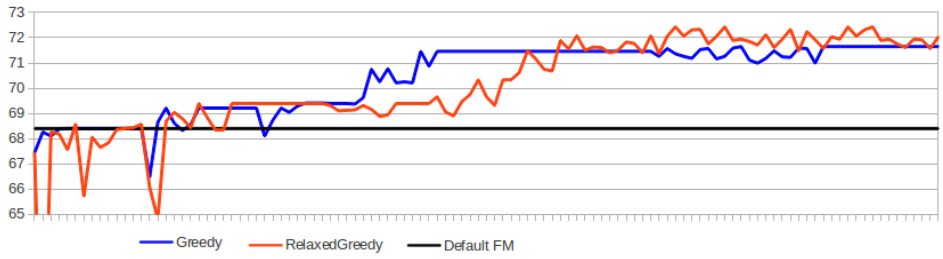



Figure C.2: Results obtained by Greedy and Relaxed Greedy in every step of the algorithms using the Slovene treebank.

### C.1.2 The Real Case: Evaluating with the CoNLL-X Shared Task Testing Data Sets

In order to find out if the algorithms overfit or underfit the performance, and also whether our methods are useful or not, we decided to test the obtained feature model with the real testing data set used in the CoNLL-X Shared Task, the Table C.2 shows the results obtained.

Most of the differences are indeed statistically significant comparing with the default models results. According to McNemar's test[3] we get significant improvements for Dutch, Slovene, Spanish and Turkish while the ones obtained for Arabic and Swedish are not better enough. Moreover, for the languages in which we have statistically significant improvements, these ones are for $p<0.01$ and for $p<0.05$ and the Z value varies from 2.687 in the case of Spanish to 12.452 in the case of Turkish. Taking into account the size of the testing data sets these results are very good.

| Language | DefaultFM | Greedy | Relaxed Greedy |
|---|---|---|---|
| Arabic | 64.93 | **66.01** | 65.71 |
| Dutch | 72.63 | **77.23** | 76.89 |
| Slovene | 69.66 | **73.68** | 73.26 |
| Spanish | 78.68 | **80.00** | 79.84 |
| Swedish | 83.50 | 83.81 | **83.85** |
| Turkish | 56.32 | 64.11 | **64.31** |

Table C.2: Labeled attachment score with comparison to default feature model and the greedy approach for a selection of languages from the CoNLL-X shared task using the provided test set and training with the entire training set.

[3]In statistics, McNemar's test is used to determine whether two different set of results provide a marginal homogeneity or they are actually statiscally significant. In dependency parsing, it is possible to run this test automatically by making use of MaltEval (Nilsson and Nivre, 2008)

Comparing the Greedy algorithm and the Relaxed Greedy algorithm we can conclude that the Greedy algorithm (which is much more faster) is more capable of providing a competitive feature model for the real case (in which the user would need to parse sentences that are not included neither in the test set nor in the training set) because the Relaxed Greedy models seem to be overfitted to the test set in most of the cases. Running the McNemar's test most of the differences are not statistically significant neither for $p<0.01$ nor for $p<0.05$, but for the Slovene treebank there is a statistically significant difference for $p<0.05$ with a Z value of 2.171, taking into account that the test sets are so small. The better results given over most of languages nevertheless strongly suggests that the simple Greedy algorithm is more accurate and it does not underfit the performance.

These results led us to think that we should consider more conservative criteria for accepting improvements during feature selection. Therefore, in the following section we show a much more informative approach, a K-Fold cross validation experiment for the Greedy algorithm, which provides better results in the present experiment and it is the only one that provides a statistically significant difference (following McNemar's test) using the Slovene treebank. Moreover, being the Greedy algorithm much more faster, we can conclude that our heuristics to reduce the number of tests were definitely correct.

## C.2 5-Fold Cross Experiment

We decided to carry out a 5-fold cross validation experiment to be included in the validation step of the Greedy algorithm due to the results obtained in the real case with Greedy and Relaxed Greedy and taking into account that one of the best ways of estimating the generalization performance of a model trained on a subset of features is to use cross-validation, as shown in (John, Kohavi, and Pfleger, 1994) for wrapper based feature selection.

We divided the corpus in 5 folds because in order to have similar number of sentences in the folds as we had in the previous experiments, when we divided the corpus in 80% for training and 20% for testing.

It is well known that there are various ways of extracting the folds from the training corpora. For the present experiment and in order to get a more complex and interesting comparison we try two different approaches:

**I.** Extracting the sentences in an iterative way, by doing a simple split, firstly the sentences for fold 1, then sentences for fold 2 and so on, this approach lead to very homogeneous corpora.

**II.** A pseudo randomize selection of sentences which provides much more heterogeneous folds.

We could come up with the following hypotheses: we could expect that the simple split selection of sentences will underfit the performance and we could also expect that the pseudo randomize selection will provide much better and informative results.

We also decided to implement three different criteria in order to decide if a feature is worth to be included in the final feature model or not:

(i) Considering that the average LAS over all folds must beat the result of the best feature model so far.

(ii) Considering that the majority of folds (in this case 3 of 5) must beat the result of the best feature model so far.

(iii) Considering that all the folds must beat the result of the best feature model so far.

Therefore, we could come up with the following hypotheses regardless or whether we use the simple split selection of sentences or the pseudo-randomize selection of sentences:

1. We could expect that (i) and (ii) will provide similar results, and it seems that both of them will neither underfit nor overfit the performance.

2. We could also expect that (iii) is going to underfit the performance in most of the cases.

In the following Subsections we show a set of experiments in which we see whether our hypotheses are corroborated or falsified.

### C.2.1 Simple Split Selection of Sentences

These results are shown in Table C.3. The simple split selection of sentences only provides improvements for Slovene and Turkish for the *average* and the *majority* criteria, producing 70.24 LAS in the case of Slovene and 66.00 LAS in the case of Turkish. It seems that this selection of sentences is not very representative of the data set and this fact misleads the results when considering 5-fold cross validation.

Surprisingly, the *average* and the *majority* criteria came up with the same feature set and in the real case (training with the whole training set and testing with the test set) they get 73.52 LAS in the case of Slovene, and 64.45 LAS in the case of Turkish. These results compared with the ones that we get with the simple Greedy step wise approach are better for Turkish (+0.3) and worse for Slovene (-0.2). These differences are not statistically significant according to McNemar's test, neither for $p<0.01$ nor for $p<0.05$.

Figure C.3 shows the results of the 5 folds, with *average* criterion, simple split selection of sentences and the Slovene corpus. In the Figure, is

| Language | DefaultFM | Greedy | Average | Majority | All |
|---|---|---|---|---|---|
| Arabic | 63.84 | **65.56** | 63.84 | 63.84 | 63.84 |
| Dutch | 78.02 | **82.63** | 78.02 | 78.02 | 78.02 |
| Slovene | 68.40 | **71.71** | 70.24 | 70.24 | 68.40 |
| Spanish | 76.64 | **79.38** | 76.64 | 76.64 | 76.64 |
| Swedish | 83.50 | **84.09** | 83.50 | 83.50 | 83.50 |
| Turkish | 58.29 | **66.92** | 66.00 | 66.00 | 58.29 |

Table C.3: Labeled attachment score with comparison to default feature model and the greedy approach for a selection of languages from the CoNLL-X shared task, reporting the results of the 5-fold cross validation with single-split selection of sentences.

evidenced the conservative behavior of this selection of sentences, in which most of the folds can not obtain better results than the default models in most of the iterations. Moreover, the Fold 3 is always below the default results.

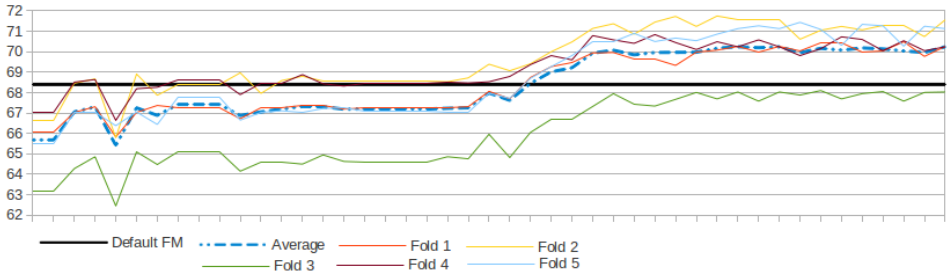



Figure C.3: Results obtained by the 5 fold cross experiments with simple split selection of sentences in every step of the algorithm using the Slovene treebank.

### C.2.2 Pseudo Randomize Selection of Sentences

We believe that the pseudo randomize selection of sentences producing a stratified sampling is much more representative of the real case and the expected results that a final user could get, here we want to demonstrate it. Our method provides the results of Table C.4. In the Table we also show the results of the Greedy algorithm without making use of the 5-fold cross validation, which can be understood as the same process but considering only the outcomes of one of the folds with pseudo randomize selection of sentences. Figure C.4 shows the results of the 5 folds, with *average* criterion, pseudo randomize selection of sentences and the Slovene corpus, we can see how all the folds produce high results if we compare with the simple split

selection of sentences shown in Figure C.3.

| Language | DefaultFM | Greedy | Average | Majority | All |
|---|---|---|---|---|---|
| Arabic | 63.84 | 65.56 | 66.44 | **66.62** | 65.33 |
| Dutch | 78.02 | **82.63** | 82.32 | 82.29 | 81.42 |
| Slovene | 68.40 | 71.71 | **72.00** | **72.00** | 69.46 |
| Spanish | 76.64 | **79.38** | 79.29 | 79.29 | 76.64 |
| Swedish | 83.50 | **84.09** | 83.50 | 83.50 | 83.50 |
| Turkish | 58.29 | 66.92 | 67.11 | 67.01 | **67.37** |

Table C.4: Labeled attachment score with comparison to default feature model and the greedy approach for a selection of languages from the CoNLL-X shared task, reporting the results of the 5-fold cross validation with stratified sampling selection.

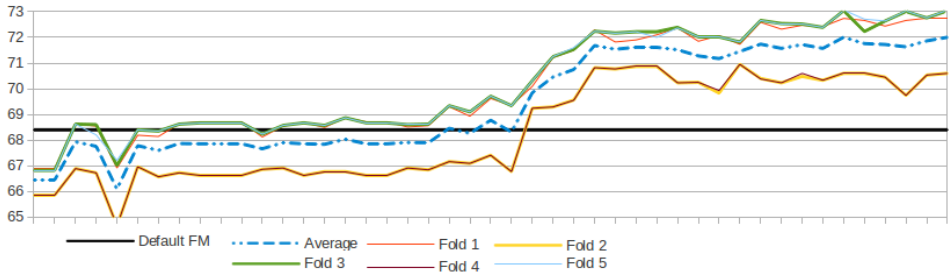



Figure C.4: Results obtained by the 5 fold cross experiments with pseudo randomize selection of sentences in every step of the algorithm using the Slovene treebank.

As we can see the results of the 5 fold cross validation strategy are much more informative, intuitively, we can rely more in the feature models obtained during the process because they have been tested over 5 different folds and represent the real case in an accurate way. In order to demonstrate this fact, we set up Table C.5 which shows the results of the obtained feature model when we test them with the test set of the CoNLL-X Shared Task.

As observed in Table C.5, the 5-fold cross validation produces higher results for Arabic, Spanish and Turkish, while the simple Greedy algorithm produces better results in the other 3. The *All* criterion seems to be very restrictive because it leads to underfitting, however, *average* and *majority* produce similar and robust results.

Nevertheless, the differences for Slovene are statistically significant according to McNemar's test in favor for the Greedy algorithm for $p<0.01$ and for $p<0.05$. But, the differences for the Turkish algorithm are statistically significant in favor of the 5-fold cross experiment (for the three cases) running McNemar's test only for $p<0.05$ and a Z value of 2.296. The rest of

| Language | DefaultFM | Greedy | Average | Majority | All |
|---|---|---|---|---|---|
| Arabic | 64.93 | 66.01 | 66.21 | **66.27** | 65.61 |
| Dutch | 72.63 | **77.23** | 76.97 | 76.39 | 75.73 |
| Slovene | 69.66 | **73.68** | 73.32 | 73.32 | 71.64 |
| Spanish | 78.68 | 80.00 | **80.46** | **80.46** | 78.68 |
| Swedish | 83.50 | **83.81** | 83.59 | 83.59 | 83.59 |
| Turkish | 56.32 | 64.11 | 64.85 | **65.01** | 64.99 |

Table C.5: Labeled attachment score with comparison to default feature model and the greedy approach for a selection of languages from the CoNLL-X shared task, reporting the results of the 5-fold cross validation, making use of the training and test set of the CoNLL-X Shared Task

the differences are not significant.

We can conclude that the Greedy algorithm by itself can provide the same good results as we can get with a much more informative experiment, in this case: K-Fold cross validation. Nonetheless, it seems worth to carry out both experiments because in some cases we can find statistically significant improvements when we check over 5 different divisions of the corpus (or folds) and vice versa.

Comparing the results of the simple split selection of sentences (shown in Section C.3) for Slovene and Turkish (which are the ones that provide improvements) with the corresponding outputs produced by the pseudo randomize selection of sentences by running McNemmar's test. We get a statistically significant different in favor of the pseudo randomize selection for $p<0.05$. Therefore, we can also conclude that the results produced by the pseudo randomize selection are not overfitted and the ones produced by the simple split selection of sentences are underfitted by a misleading selection of sentences.

## C.3 Conclusions of the Appendix

We have demonstrated that different criteria for automatic feature selection in the case of transition based dependency parsing can be accomplished successfully and produce variant results in the final performance. Moreover, both search algorithm presented produce consistent and often very substantial improvements over the default settings for MaltParser using different validation procedures, such as K-fold cross validation or simple slight improvements over an unseen division of the corpus. According to our results and taking into account that all of our results are inherently based on the same Greedy algorithm, we believe that it does not matter much whether we use different validation procedures or different decision rules but, as shown

in the experiments, it is always worth to check several of them in order to get the most convincing outcome.

# Appendix D

# Optimizing the Multiplanar Parsers with MaltOptimizer

In this Appendix we present two different sets of experiments with corpora from the CoNLL-X Shared Task (Buchholz and Marsi, 2006). We forced MaltOptimizer (by using the possible interaction between phases) to use Planar arc-eager and 2-Planar arc-eager (Gómez-Rodríguez and Nivre, 2010) as selected parsing algorithms in order to run a full feature selection for these two parsers and to observe how far we can go with the feature selection for a single algorithm in spite of selecting the algorithms automatically.

The Multiplanar parsers were not selected as best parsers for the corpora in which we carried out experiments with MaltOptimizer shown in Table 5.3, due to the action of the rest of the algotithms, which provide normally higher results. However, the experiments shown in this Appendix are interesting because we are mainly showing that we can provide an optimization for all the parsing algorithms included in MaltParser, moreover a researcher may be interested in optimizing a single parsing algorithm as a matter of comparison.

Table D.1 shows the results for all the MaltOptimizer phases. Default and Phase 1 columns present the outcomes of the first phase in which Nivre arc-eager was selected as default parsing algorithm. Phase 2 columns, is divided in 2, in which we show the results of Planar and 2-Planar.[1] Finally, in Phase 3, we show the results of feature selection again for Planar and 2-Planar, interacting with the possibility that MaltOptimizer provides, stopping the process between phases. For phase 3, when the training corpora has non zero non-projective arcs/trees we run Planar arc-eager with pseudo-projective parsing, otherwise, we run it with default settings.

We can note that the performance improved substantially over all the se-

[1]For 2-Planar, as we saw in Section 5.2.2 MaltOptimizer selects between the 2-Planar options (`reduceonswitch` and `planar root handling`) making it believe that 2-Planar is the best parsing algorithm for the data.

| | | | Phase 2 | | Phase 3 | |
|---|---|---|---|---|---|---|
| **Language** | **Default** | **Phase 1** | **Planar** | **2-Planar** | **Planar** | **2-Planar** |
| Arabic | 63.02 | 63.03 | 62.81 | 63.42 | 65.53 | 64.94 |
| Bulgarian | 83.19 | 83.19 | 82.89 | 84.09 | 83.55 | 84.09 |
| Chinese | 84.14 | 84.14 | 81.66 | 82.79 | 83.63 | 83.54 |
| Czech | 69.85 | 70.24 | 70.34 | 70.45 | 75.60 | 74.76 |
| Danish | 81.01 | 81.01 | 80.86 | 81.18 | 82.08 | 82.75 |
| Dutch | 74.77 | 74.77 | 76.55 | 76.81 | 76.55 | 81.41 |
| German | 82.36 | 82.36 | 81.34 | 82.28 | 84.64 | 84.84 |
| Japanese | 89.70 | 89.70 | 86.62 | 88.19 | 87.95 | 89.79 |
| Portuguese | 84.11 | 84.31 | 84.06 | 84.19 | 84.06 | 86.10 |
| Slovene | 66.08 | 66.52 | 65.94 | 66.43 | 69.72 | 70.13 |
| Spanish | 76.45 | 76.45 | 75.69 | 76.52 | 78.29 | 79.15 |
| Swedish | 83.34 | 83.34 | 82.38 | 82.83 | 83.67 | 83.78 |
| Turkish | 57.79 | 57.79 | 55.94 | 56.08 | 64.44 | 64.00 |

Table D.1: Labeled attachment score per phase and with comparison to default settings for the training sets from the CoNLL-X shared task.

lected corpora. However, for some data sets we got a much more substantial improvement, such as Turkish or Slovene, than in the other corpora.

Note that the results for Czech are calculated with half of the training corpus due to the impossibility of creating in Liblinear an array bigger than $2^{32}$, which occurred with Czech, Planar arc-eager and pseudo-projective with `head` option. This fact did not happen in the experiments shown in Chapter 5 because Planar arc-eager + pseudo-projective (`head`) were not selected as best parsing algorithm.

It is also worth noting that 2-Planar arc-eager normally provides better results, both in default settings (Phase 2) and in the optimized version (Phase 3), than Planar arc-eager. However, in some cases this is not the case, and the feature selection for Planar arc-eager reaches a higher attachment score.

# Appendix E

# Lexicon of Negation Cues.

In this Appendix we show the static lexicon used by the system presented to the *SEM Shared Task. The system is explained and discussed in Section 6.2.6.

| | | | | |
|---|---|---|---|---|
| absence | breathless | breathlessness | by no means | cannot |
| careless | carelessness | colourless | disapprobation | disconnected |
| disfavour | dislike | displeasure | dissatisfied | distasteful |
| except | fail | fearless | godless | harmless |
| helpless | helplessly | hopeless | immaterial | immutable |
| impassable | impatience | impatient | impatiently | impenetrable |
| impossible | improper | imprudent | inadequate | inadmissable |
| inadvertently | inconceivable | inconclusive | inconvenient | incredible |
| incredulously | indescribably | indiscreet | inexplicable | infinite |
| infrequent | inhospitable | inscrutable | insensible | insensibly |
| insufferable | interminable | intolerable | invisible | irregular |
| irrelevant | irreproachable | irresolute | irretrievably | irrevocable |
| lifeless | motionless | needless | neglected | neither |
| neither nor | never | no | no more | no nor |
| nobody | noiselessly | none | nor | not |
| nothing | nothing at all | nowhere | n't | on the contrary |
| powerless | prevent | purposeless | rather than | refused |
| restlessly | sapless | save | shelterless | unable |
| unambitious | unarmed | unbroken | unbrushed | unburned |
| uncanny | uncertain | unclean | uncomfortably | uncommon |
| uncommonly | unconcerned | unconcernedly | unconscious | uncontrollable |
| unconventional | uncurtained | undeceived | undeniable | undoubtedly |
| uneasiness | uneasy | uneducated | unemotional | unexpected |
| unexplored | unfair | unfairly | unfortunate | unfortunately |
| unfounded | unfruitful | unfurnished | unhappy | unimaginative |
| uninteresting | unjustifiable | unkempt | unknown | unlike |
| unlikely | unlimited | unmarried | unmistakable | unmitigated |
| unnatural | unnecessary | unoccupied | unpleasant | unpractical |
| unsafe | unseen | unshaven | unsigned | untenanted |
| untimely | unusual | unwarlike | useless | windless |
| without | worthless | | | |

# Appendix F

# Publications

In this Appendix we show the publications that were published during the development of the present thesis.

## F.1 Analyzing Dependency Analysis

1. Miguel Ballesteros, Jesús Herrera, Virginia Francisco and Pablo Gervás. 2010. Improving Parsing Accuracy for Spanish using Maltparser. Revista de la Sociedad Española para el Procesamiento del Lenguaje natural (SEPLN) number 44. 83-90.

2. Miguel Ballesteros, Jesús Herrera, Virginia Francisco and Pablo Gervás. 2012. Analyzing the CoNLL-X Shared Task from a Sentence Accuracy Perspective. Revista de la Sociedad Española para el Procesamiento del Lenguaje natural (SEPLN) number 48. 29-34.

3. Miguel Ballesteros, Jesús Herrera, Virginia Francisco and Pablo Gervás. 2012. Are the existing training corpora unnecessarily large?. Revista de la Sociedad Española para el Procesamiento del Lenguaje natural (SEPLN) number 48. Pages 21-27.

## F.2 Enhancing Dependency Analysis

1. Miguel Ballesteros, Jesús Herrera, Virginia Francisco and Pablo Gervás. 2010. A Feasibility Study on Low Level Techniques for Improving Parsing Accuracy for Spanish Using Maltparser. In Proceedings of the 6th Hellenic Conference on Artificial Intelligence (SETN 2010). Athens, Greece. Springer LNAI 6040. 39-48.

2. Miguel Ballesteros, Jesús Herrera, Virginia Francisco and Pablo Gervás. 2010. Towards an N-Version Dependency Parser. In Proceedings of

the Text Speech and Dialogue Conference (TSD 2010). Brno, Czech Republic. Springer LNAI 6231. 43-50.

3. Miguel Ballesteros, Jesús Herrera, Virginia Francisco and Pablo Gervás. 2010. Giving Shape to an N-Version Dependency Parser. In Proceedings of the International Conference on Knowledge Discovery and Information Retrieval (IC3K-KDIR 2010). Valencia, Spain. INSTICC. 336-341.

4. Miguel Ballesteros, Jesús Herrera, Virginia Francisco and Pablo Gervás. 2012. Enhancing Dependency Analysis by Combining Specific Dependency Parsers. Hybrid Intelligent Systems: Concepts and Applications (Book Chapter). iConcept Press, Ltd. (Pages 16)

## F.3 Optimizing Dependency Analysis

1. Miguel Ballesteros and Joakim Nivre. 2012. MaltOptimizer: An Optimization Tool for MaltParser. In Proceedings of the System Demonstration Session of the Thirteenth Conference of the European Chapter of the Association for Computational Linguistics (EACL 2012). Avignon, France. ACL Onthology. 58-62.

2. Miguel Ballesteros and Joakim Nivre. 2012. MaltOptimizer: A System for MaltParser Optimization. In Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC 2012), Istanbul, Turkey. 2757-2763.

3. Miguel Ballesteros, Carlos Gómez-Rodríguez and Joakim Nivre. 2012. Optimizing Planar and 2-Planar Parsers with MaltOptimizer. Revista de la Sociedad Española para el Procesamiento del Lenguaje natural. (SEPLN) number 49. (Pages 8)

## F.4 Applying Dependency Analysis

1. Miguel Ballesteros, Susana Bautista and Pablo Gervás. 2010. Text Simplification Using Dependency Parsing for Spanish. In Proceedings of the International Conference on Knowledge Discovery and Information Retrieval (IC3K-KDIR 2010). Valencia, Spain. INSTICC. 330-335.

2. Pablo Gervás and Miguel Ballesteros 2011. A Proposal for a Spanish Surface Realisation Shared Task. In Proceedings of the 13th European Workshop on Natural Language Generation. Nancy France. ACL Onthology. 212-216.

3. Miguel Ballesteros, Virginia Francisco, Alberto Díaz, Jesús Herrera and Pablo Gervás. 2012. Inferring the Scope of Negation in Biomedical Documents. 13th International Conference on Intelligent Text Processing and Computational Linguistics (CICLING 2012), New Delhi, India. Springer LNCS 7181. 363-375.

4. Miguel Ballesteros, Alberto Díaz, Virginia Francisco, Pablo Gervás, Jorge Carrillo de Albornoz and Laura Plaza. 2012. UCM-2: a Rule-Based Approach to Infer the Scope of Negation via Dependency Parsing. *SEM Shared Task 2012. Resolving the Scope and Focus of Negation, (*SEM 2012) Montreal, Canada. ACL Onthology. 288-293.

# Appendix G

# Research Stays

During the period in which I developed my PhD thesis, I visited the Department of Linguistic and Philology in Uppsala University twice. I was supervised by the Professor of Computational Linguistics **Joakim Nivre**:

1. Uppsala University, Sweden. Supervised by Joakim Nivre. 3rd October 2011 - 23rd December 2011.

2. Uppsala University, Sweden. Supervised by Joakim Nivre. 19th March 2012 - 19th April 2012.

In these research stays, we studied and developed the research explained in Section 4.2 and the entire Chapter 5. Moreover, the papers shown in Section A.3 were written, published and produced as a result of the two stays.

# Appendix H

# Invited Talks and Seminars

I was invited to give the following talks in which I presented works related to my thesis:

1. UCNLG+Eval - The 4th UCNLG Workshop: Language Generation and Evaluation,[1] Edinburgh, United Kingdom. Invited by Anja Belz.
   31st July 2011.
   Talk Title: *A Proposal for a Spanish Surface Realization Shared Task.*

2. Seminars of the Department of Linguistics and Philology, Uppsala University, Sweden.[2] Invited by Joakim Nivre.
   14th October 2011.
   Talk Title: *Enhancing and Applying Dependency Analysis in Different Topics.*

3. Department of Computer Science, Stockholm University, Kista, Sweden.[3] Invited by Hercules Dalianis.
   20th October 2011.
   Talk Title: *Enhancing and Applying Dependency Analysis in Different Topics: Negation, Speculation and Text Simplification.*

4. Seminars of the Department of Linguistics and Philology, Uppsala University, Sweden.[4] Invited by Joakim Nivre.
   30th March 2012.
   Talk Title: *MaltOptimizer: A System for MaltParser Optimization.*

[1] http://www.nltg.brighton.ac.uk/ucnlg/ucnlg11/programme.html
[2] http://stp.lingfil.uu.se/ joerg/seminars.html
[3] http://dash.dsv.su.se/2011/10/20/it-for-health-seminar-miguel-ballesteros-enhancing-and-applying-dependency-analysis-in-different-topics-negation-speculation-and-text-simplification/
[4] http://stp.lingfil.uu.se/ joerg/seminars.html

## References

Abeillé, Anne, editor. 2003. *Treebanks: Building and Using Parsed Corpora*, volume 20 of *Text, Speech and Language Technology*. Kluwer Academic Publishers, Dordrecht.

Aduriz, I., M. J. Aranzabe, J. M. Arriola, A. Atutxa, A. Díaz de Ilarraza, A. Garmendia, and M. Oronoz. 2003. Construction of a Basque dependency treebank. In *Proceedings of the Third Workshop on Treebanks and Linguistic Theories (TLT)*, pages 201–204, Växjö, Sweden.

Afonso, Susana, Eckhard Bick, Renato Haber, and Diana Santos. 2002. Floresta sintá(c)tica: A treebank for Portuguese. In *Third International Conference on Language Resources and Evaluation (LREC 2002) (Las Palmas de Gran Canaria, Spain)*.

Agarwal, Shashank and Hong Yu. 2010. Biomedical negation scope detection with conditional random fields. *Journal of the American Medical Informatics Associations*.

Anguiano, Enrique Henestroza and Marie Candito. 2011. Parse correction with specialized models for difficult attachment types. In *EMNLP*, pages 1222–1233.

Atalay, Nart B., Kemal Oflazer, and Bilge Say. 2003. The annotation process in the Turkish treebank. In *LINC2003*.

Attardi, Giuseppe. 2006. Experiments with a multilanguage non-projective dependency parser. In *CoNLL-X '06: Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 166–170, Morristown, NJ, USA. Association for Computational Linguistics.

Ballesteros, Miguel, Susana Bautista, and Pablo Gervás. 2010. Text Simplification Using Dependency Parsing for Spanish. In *KDIR–Knowledge Discovery and Information Retrieval 2010*, pages 330–335, Valencia, Spain. INSTICC.

Ballesteros, Miguel, Alberto Díaz, Virginia Francisco, Pablo Gervás, Jorge Carrillo de Albornoz, and Laura Plaza. 2012. Ucm-2: a rule-based approach to infer the scope of negation via dependency parsing. In **SEM Shared Task 2012. Resolving the Scope and Focus of Negation*, Montreal, Canada.

Ballesteros, Miguel and Joakim Nivre. 2012. MaltOptimizer: An Optimization Tool for MaltParser. In *Proceedings of the System Demonstration Session of the Thirteenth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.

Basile, Valerio, Johan Bos, Kilian Evang, and Noortje Venhuizen. 2012. Ugroningen: Negation detection with discourse representation structures. In **SEM Shared Task 2012. Resolving the Scope and Focus of Negation*, Montreal, Canada.

Bick, Eckhard. 2000. *The Parsing System "Palavras". Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework.* University of Arhus.

Bick, Eckhard. 2006. Lingpars, a linguistically inspired, language-independent machine learner for dependency treebanks. In *CoNLL-X '06: Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 171–175, Morristown, NJ, USA. Association for Computational Linguistics.

Böhmová, Alena, Jan Hajič, Eva Hajičová, and Barbora Hladká. 2003. The PDT: a 3-level annotation scenario. In Abeillé (Abeillé, 2003), chapter 7.

Bohnet, Bernd. 2009. Efficient parsing of syntactic and semantic dependency structures. In *Proceedings of CoNLL-2009*, pages 67–72.

Bohnet, Bernd. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *COLING*, pages 89–97.

Bohnet, Bernd and Jonas Kuhn. 2012. The best of bothworlds - a graph-based completion model for transition-based parsers. In *EACL*.

Bohnet, Bernd, Simon Mille, Benoît Favre, and Leo Wanner. 2011. <stumaba >: From deep representation to surface. In *Proceedings of the Generation Challenges Session at the 13th European Workshop on Natural Language Generation*, pages 232–235, Nancy, France, September. Association for Computational Linguistics.

Bosco, Cristina, Simonetta Montemagni, Alessandro Mazzei, Vincenzo Lombardo, Felice dell'Orletta, Alessandro Lenci, Leonardo Lesmo, Giuseppe Attardi, Maria Simi, Alberto Lavelli, Johan Hall, Jens Nilsson, and Joakim Nivre. 2010. Comparing the influence of different treebank annotations on dependency parsing. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC).*

Brants, Sabine, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The tiger treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories (TLT).*

Buchholz, Sabine and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 149–164.

Canisius, Sander, Toine Bogers, Antal Van Den Bosch, and Jeroen Geertzen. 2006. Dependency parsing by inference over high-recall dependency predictions. In *CoNLL-X '06: Proceedings of the Tenth Conference on Computational Natural Language Learning*.

Canning, Yvonne. 2000. Cohesive simplification of newspaper text for aphasic readers. In *3rd annual CLUK Doctoral Research Colloquium*.

Carreras, Xavier. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 957–961.

Carreras, Xavier, Michael Collins, and Terry Koo. 2008. Tag, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning (CoNLL)*, pages 9–16.

Carreras, Xavier, Lluís Màrquez, and Jorge Castro. 2005. Filtering-ranking perceptron learning for partial parsing. *Machine Learning*, 60(1-3):41–71, September.

Carreras, Xavier, Mihai Surdeanu, and Lluís Màrquez. 2006. Projective dependency parsing with perceptron. In *CoNLL-X '06: Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 181–185, Morristown, NJ, USA. Association for Computational Linguistics.

Carrillo de Albornoz, Jorge, Laura Plaza, Alberto Díaz, and Miguel Ballesteros. 2012. Ucm-i: A rule-based syntactic approach for resolving the scope of negation. In **SEM Shared Task 2012. Resolving the Scope and Focus of Negation*, Montreal, Canada.

Carrillo de Albornoz, Jorge, Laura Plaza, Pablo Gervás, and Alberto Díaz. 2011. A joint model of feature mining and sentiment analysis for product review rating. In *Proceedings of the 33rd European Conference on Information Retrieval (ECIR 2011)*.

Caseli, Helena M., Tiago F. Pereira, Lucia Specia, Thiago A. S. Pardo, Caroline Gasperin, and Sandra M.Aluisio. 2009. Building a Brazilian Portuguese Parallel Corpus of Original and Simplified Texts. In *In Proceedings of CICLing*.

Cassel, Sofia. 2009. Maltparser and liblinear: Transition-based dependency parsing with linear classification for feature model optimization. Master's thesis, Uppsala University, Uppsala, Sweden.

Çetinoglu, Özlem, Anton Bryl, Jennifer Foster, and Josef van Genabith. 2011. Improving dependency label accuracy using statistical post-editing: A cross-framework study. In *DEPLING*.

Chandrasekar, R., Christine Doran, and B. Srinivas. 1996. Motivations and methods for text simplification. In *Proceedings of the Sixteenth International Conference on Computational Linguistics (COLING '96)*, pages 1041–1044.

Chandrasekar, R. and B. Srinivas. 1997. Automatic induction of rules for text simplification. *Knowledge-Based Systems*, 10.

Chang, Chih-Chung and Chih-Jen Lin, 2001. *LIBSVM: A Library for Support Vector Machines*. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

Chang, Ming Wei, Quang Do, and Dan Roth. 2006. A pipeline model for bottom-up dependency parsing. In *Journal of the School of Engineering, The University of Tokyo*, pages 15–40.

Chapman, Wendy W., Will Bridewell, Paul Hanbury, Gregory F. Cooper, and Bruce G. Buchanan. 2001. A simple algorithm for identifying negated findings and diseases in discharge summaries. *J Biomed Inform*, 2001:34–301.

Chapman, Wendy W., Will Bridewell, Paul Hanbury, Gregory F. Cooper, Bruce G. Buchanan, Wendy W. Chapman Phd, Will Bridewell Bs, Paul Hanbury Bs, Gregory F. Cooper Md Phd, and Bruce G. Buchanan Phd. 2002. Evaluation of negation phrases in narrative clinical reports.

Chen, Keh-Jiann, Chi-Ching Luo, Ming-Chung Chang, Feng-Yi Chen, Chao-Jan Chen, Chu-Ren Huang, and Zhao-Ming Gao. 2003. Sinica treebank: Design criteria, representational issues and implementation. In Abeillé (Abeillé, 2003), chapter 13, pages 231–248.

Chen, Wenliang, Jun'ichi Kazama, Kiyotaka Uchimoto, and Kentaro Torisawa. 2009. Improving dependency parsing with subtrees from auto–parsed data. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2*, EMNLP '09, pages 570–579, Morristown, NJ, USA. Association for Computational Linguistics.

Cheng, Yuchang, Masayuki Asahara, and Yuji Matsumoto. 2006. Multilingual dependency parsing at NAIST. In *CoNLL-X '06: Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 191–195, Morristown, NJ, USA. Association for Computational Linguistics.

Civit, M. and M. M. Antònín. 2002. Design principles for a spanish treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories(TLT02)*, Sozopol, Bulgaria.

Civit, Montserrat, M$^{a}$ Antònia Martí, Borja Navarro, Núria Bufi, Belén Fernández, and Raquel Marcos. 2003. Issues in the syntactic annotation of Cast3LB. In *LINC2003*.

Corston-Oliver, Simon and Anthony Aue. 2006. Dependency parsing with reference to slovene, spanish and swedish. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 196–200, Stroudsburg, PA, USA. Association for Computational Linguistics.

Cortes, Corinna and Vladimir Vapnik. 1995. Support-vector networks. *Machine Learning*, 20(3):273–297.

Councill, Isaac G., Ryan McDonald, and Leonid Velikovich. 2010. What's great and what's not: learning to classify the scope of negation for improved sentiment analysis. In *Proceedings of the Workshop on Negation and Speculation in Natural Language Processing*, NeSp-NLP '10, pages 51–59, Stroudsburg, PA, USA. Association for Computational Linguistics.

Covington, Michael A. 1984. *Syntactic Theory in the High Middle Ages*. Cambridge University Press.

Covington, Michael A. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102.

Crammer, Koby, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585.

Csendes, Dóra, János Csirik, Tibor Gyimóthy, and András Kocsor. 2005. The szeged treebank. In *Proceedings of the 8th international conference on Text, Speech and Dialogue*, TSD'05, pages 123–131, Berlin, Heidelberg. Springer-Verlag.

Cui, Hang, Renxu Sun, Keya Li, Min yen Kan, and Tat seng Chua. 2005. Question answering passage retrieval using dependency relations. In *In SIGIR 2005*, pages 400–407. ACM Press.

Culotta, Aron and Jeffrey Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 423–429, Barcelona, Spain, July.

Daelemans, Walter, Véronique Hoste, Fien De Meulder, and Bart Naudts. 2003. Combined optimization of feature selection and algorithm parameters in machine learning of language. In *Proceedings of the 14th European Conference on Machine Learning (ECML)*, pages 84–95.

Daelemans, Walter, Mirella Lapata, and Lluís Màrquez, editors. 2012. *EACL 2012, 13th Conference of the European Chapter of the Association for Computational Linguistics, Avignon, France, April 23-27, 2012*. The Association for Computer Linguistics.

Das, Abhimanyu and David Kempe. 2011. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 1057–1064.

Devlin, S. and J.I. Tait, 1998. *Linguist Databases*, chapter The use of a Psycholinguistic database in the Simplification of Text for Aphasic Readers, pages 161–173. CSLI.

Devlin, Siobhan and Gary Unthank. 2006. Helping aphasic people process online information. In *Assets '06: Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility*, pages 225–226, New York, NY, USA. ACM.

Doraisamy, Shyamala, Shahram Golzari, Noris Mohd. Norowi, Md Nasir Sulaiman, and Nur Izura Udzir. 2008. A study on feature selection and classification techniques for automatic genre classification of traditional malay music. In *Proceedings of the Ninth International Conference on Music Information Retrieval (ISMIR)*, pages 331–336.

Dreyer, Markus, David A. Smith, and Noah A. Smith. 2006. Vine parsing and minimum risk reranking for speed and precision. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 201–205, Stroudsburg, PA, USA. Association for Computational Linguistics.

Džeroski, Sašo, Tomaž Erjavec, Nina Ledinek, Petr Pajas, Zdenek Žabokrtsky, and Andreja Žele. 2006. Towards a Slovene dependency treebank. In *In Proc. Int. Conf. on Language Resources and Evaluation (LREC.*

Eisner, Jason M. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pages 340–345.

Fan, R.-E., K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Farkas, Richárd, Veronika Vincze, and Helmut Schmid. 2012. Dependency parsing of hungarian: Baseline results and challenges. In *EACL*, pages 55–65.

Foster, Jennifer, Özlem Çetinoglu, Joachim Wagner, Joseph Le Roux, Stephen Hogan, Joakim Nivre, Deirdre Hogan, and Josef van Genabith. 2011. #hardtoparse: Pos tagging and parsing the twitterverse. In *Analyzing Microtext*.

Goldberg, Yoav and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *NAACL*.

Gómez-Rodríguez, Carlos and Daniel Fernández-González. 2012. Dependency parsing with undirected graphs. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 66–76, Avignon, France, April. Association for Computational Linguistics.

Gómez-Rodríguez, Carlos and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1492–1501.

Guyon, Isabelle and André Elisseeff. 2003. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182.

Hajič, Jan, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the 13th Conference on Computational Natural Language Learning (CoNLL-2009), June 4-5*, Boulder, Colorado, USA.

Hajič, Jan, Otakar Smrž, Petr Zemánek, Jan Šnaidauf, and Emanuel Beška. 2004. Prague Arabic dependency treebank: Development in data and tools. pages 110–117.

Hall, Johan, Jens Nilsson, Joakim Nivre, Gülsen Eryiğit, Beáta Megyesi, Mattias Nilsson, and Markus Saers. 2007. Single malt or blended? A study in multilingual parser optimization. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 933–939.

Hays, David G. 1964. Dependency theory: A formalism and some observations. *Language*, 40:511–525.

Herrera, Jesús and Pablo Gervás. 2008. Towards a Dependency Parser for Greek Using a Small Training Data Set. *Journal of the Spanish Society for Natural Language Processing (SEPLN)*, 41:29–36.

Herrera, Jesús, Anselmo Peñas, and Felisa Verdejo. 2005. Textual entailment recognition based on dependency analysis and *wordnet*. In *MLCW*, pages 231–239.

Horn, Laurence R. 1989. *A natural history of negation / Laurence R. Horn.* University of Chicago Press, Chicago :.

Huang, Y. and H. J. Lowe. 2007. A Novel Hybrid Approach to Automated Negation Detection in Clinical Radiology Reports. *Journal of the American Medical Informatics Association*, 14(3):304–311, May.

Johansson, Richard and Pierre Nugues. 2006. Investigating multilingual dependency parsing. In *CoNLL-X '06: Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 206–210, Morristown, NJ, USA. Association for Computational Linguistics.

John, George H., Ron Kohavi, and Karl Pfleger. 1994. Irrelevant Features and the Subset Selection Problem. In *International Conference on Machine Learning*, pages 121–129.

Kawata, Yasuhiro and Julia Bartels. 2000. Stylebook for the Japanese treebank in VERBMOBIL. Verbmobil-Report 240, Seminar für Sprachwissenschaft, Universität Tübingen.

Kim, J. D., T. Ohta, Y. Tateisi, and J. Tsujii. 2003. GENIA corpus: a semantically annotated corpus for bio-textmining. *Bioinformatics*, 19(suppl 1):i180–i182, July.

Klebanov, Beata Beigman, Kevin Knight, and Daniel Marcu. 2004. Text simplification for information-seeking applications. In *On the Move to Meaningful Internet Systems, Lecture Notes in Computer Science*, pages 735–747. Springer Verlag.

Klein, Dan and Christopher D. Manning. 2002. Conditional structure versus conditional estimation in NLP models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9–16.

Koo, Terry, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *EMNLP*, pages 1288–1298.

Kool, Anne, Jakub Zavrel, and Walter Daelemans. 2000. Simultaneous feature selection and parameter optimization for memory-based natural

language processing. In *Proceedings of the Tenth Belgian-Dutch Conference on Machine Learning (BENELEARN)*, pages 93–100.

Korycinski, Donna, Melba Crawford, J. Wesley Barnes, and Joydeep Ghosh. 2003. Adaptive feature selection for hyperspectral data analysis. In *Proceedings of the SPIE Conference on Image and Signal Processing for Remote Sensing IX*, pages 213–225.

Kromann, Matthias T. 2003. The Danish dependency treebank and the underlying linguistic theory. Växjö, Sweden.

Kübler, Sandra, Ryan McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Morgan and Claypool.

Lapponi, Emanuele, Erik Velldal, Lilja Øvrelid, and Jonathon Read. 2012. Uio2: Sequence-labeling negation using dependency features. In **SEM Shared Task 2012. Resolving the Scope and Focus of Negation*, Montreal, Canada.

Lin, Dekang. 1998. Dependency-based evaluation of MINIPAR. In *Proc. Workshop on the Evaluation of Parsing Systems*, Granada.

Liu, Ting, Jinshan Ma, Huijia Zhu, and Sheng Li. 2006. Dependency parsing based on dynamic local optimization. In *CoNLL-X '06: Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 211–215, Morristown, NJ, USA. Association for Computational Linguistics.

Luque, Franco M., Ariadna Quattoni, Borja Balle, and Xavier Carreras. 2012. Spectral learning for non-deterministic dependency parsing. In *EACL*, pages 409–419.

Marcus, Mitchell P., Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: the penn treebank. *Comput. Linguist.*, 19(2):313–330, June.

Martí, M. Antònia, M. Taulé, L. Màrquez, and M.Bertran. 2007. Cess-ece: A multilingual and multilevel annotated corpus. Draft Version.

Max, Aurélien. 2006. Writing for language-impaired readers. In *CICLing*, pages 567–570.

McCallum, Andrew. 2003. Efficiently inducing features of conditional random fields. In *Proceedings of the Conference on Uncertainty in AI*, pages 403–410.

McDonald, R., K. Lerman, and F. Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the*

*10th Conference on Computational Natural Language Learning (CoNLL–X)*, pages 216–220.

McDonald, Ryan, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 91–98.

McDonald, Ryan, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 216–220.

McDonald, Ryan and Joakim Nivre. 2007. Characterizing the errors of data–driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 122–131. Association for Computational Linguistics.

McDonald, Ryan and Joakim Nivre. 2011. Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(1):197–230.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 523–530.

Mel'čuk, Igor. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press.

Montemagni, S., F. Barsotti, M. Battista, N. Calzolari, O. Corazzari, A. Zampolli, F. Fanciulli, M. Massetani, R. Raffaelli, R. Basili, M. T. Pazienza, D. Saracino, F. Zanzotto, N. Mana, F. Pianesi, and R. Delmonte. 2003. Building the italian syntactic-semantic treebank. In *In Abeillé (Abeillé, 2003), chapter 11*, pages 189–210. Kluwer.

Morante, Roser. 2010. Descriptive analysis of negation cues in biomedical texts. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, Valletta, Malta, may. European Language Resources Association (ELRA).

Morante, Roser and Eduardo Blanco. 2012. Sem 2012 shared task: Resolving the scope and focus of negation. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics (*SEM 2012), Montreal, Canada.*

Morante, Roser and Walter Daelemans. 2009. A metalearning approach to processing the scope of negation. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, CoNLL '09, pages 21–29, Stroudsburg, PA, USA. Association for Computational Linguistics.

Morante, Roser and Walter Daelemans. 2012. Conandoyle-neg: Annotation of negation in conan doyle stories. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC). Istanbul, Turkey.*

Morante, Roser, Anthony Liekens, and Walter Daelemans. 2008. Learning the scope of negation in biomedical texts. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 715–724, Stroudsburg, PA, USA. Association for Computational Linguistics.

Morante, Roser and Caroline Sporleder, editors. 2010. *Proceedings of the Workshop on Negation and Speculation in Natural Language Processing*, Uppsala, Sweden, 07/2010. University of Antwerp.

Mutalik, Pradeep G., Aniruddha Deshpande, and Prakash M. Nadkarni. 2001. Use of General-purpose Negation Detection to Augment Concept Indexing of Medical Documents: A Quantitative Study using the UMLS. *Journal of the American Medical Informatics Association*, 8(6):598–609.

Navarro, B., M. Civit, M. A. Martí, B. Fernández, and R. Marcos. 2003. Syntactic, semantic and pragmatic annotation in cast3lb. In *Proceedings of the Shallow Processing of Large Corpora. A Corpus Linguistics Workshop.*, Lancaster, UK.

Nilsson, Jens, Johan Hall, and Joakim Nivre. 2005. MAMBA meets TIGER: Reconstructing a Swedish treebank from antiquity. In *Proc. of the NODALIDA Special Session on Treebanks.*

Nilsson, Jens and Joakim Nivre. 2008. Malteval: an evaluation and visualization tool for dependency parsing. In Bente Maegaard Joseph Mariani Jan Odjik Stelios Piperidis Daniel Tapias Nicoletta Calzolari (Conference Chair), Khalid Choukri, editor, *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, may. European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2008/.

Nilsson, Peter and Pierre Nugues. 2010. Automatic discovery of feature sets for dependency parsing. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 824–832.

Nivre, Joakim. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Nivre, Joakim. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*, pages 50–57.

Nivre, Joakim. 2005. Dependency grammar and dependency parsing. Technical report, Växjö University.

Nivre, Joakim. 2006. *Inductive Dependency Parsing*. Springer.

Nivre, Joakim. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34:513–553.

Nivre, Joakim. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL-IJCNLP)*, pages 351–359.

Nivre, Joakim and Johan Hall. 2010. A quick guide to MaltParser optimization. Technical report, maltparser.org.

Nivre, Joakim, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic, June. Association for Computational Linguistics.

Nivre, Joakim, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL)*, pages 49–56.

Nivre, Joakim, Johan Hall, and Jens Nilsson. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, pages 2216–2219.

Nivre, Joakim, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kbler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Nivre, Joakim, Johan Hall, Jens Nilsson, Gülsen Eryiğit, and Svetoslav Marinov. 2006a. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225.

Nivre, Joakim, Johan Hall, Jens Nilsson, Gülsen Eryiğit, and Svetoslav Marinov. 2006b. Labeled pseudo–projective dependency parsing with support vector machines. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL–X)*, pages 221–225.

Nivre, Joakim, Marco Kuhlmann, and Johan Hall. 2009. An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76.

Nivre, Joakim and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 950–958.

Nivre, Joakim and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106.

Oflazer, Kemal, Bilge Say, Dilek Zeynep Hakkani-Tür, and Gökhan Tür. 2003. Building a Turkish treebank. In Abeillé (Abeillé, 2003), chapter 15.

Pahikkala, Tapio, Antti Airola, and Tapio Salakoski. 2010. Speeding up greedy forward selection for regularized least-squares. In *The Ninth International Conference on Machine Learning and Applications*, pages 325–330.

Palomar, M., M. Civit, A. Díaz, L. Moreno, E. Bisbal, M. Aranzabe, A. Ageno, M.A. Martí, and Navarro. 2004. 3lb: Construcción de una base de datos de árboles sintáctico–semánticos para el catalán, euskera y español. In *Proceedings of the XX Conference of the Spanish Society for Natural Language Processing (SEPLN)*, pages 81–88. Sociedad Española para el Procesamiento del Lenguaje Natural.

Petersen, Sarah E. and Mari Ostendorf. 2007. Text simplification for language learners: a corpus analysis. In *In Proc. of Workshop on Speech and Language Technology for Education.*

Petrov, Slav, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440.

Petrov, Slav, Dipanjan Das, and Ryan McDonald. 2012. A universal part-of-speech tagset. In *Proceedings of the Eight International Conference*

*on Language Resources and Evaluation (LREC 12)*, Istanbul, Turkey, may. European Language Resources Association (ELRA).

Prokopidis, Prokopis, Elina Desypri, Maria Koutsombogera, Harris Papageorgiou, and Stelios Piperidis. 2005. Theoretical and Practical Issues in the Construction of a Greek Dependency Treebank. In *Proceedings of The Fourth Workshop on Treebanks and Linguistic Theories (TLT 2005), Barcelona, Spain*, pages 149–160.

Prudhvi Kosaraju, Sruthilaya Reddy Kesidi, Vinay Bhargav Reddy Ainavolu and Puneeth Kukkadapu. 2010. Experiments on indian language dependency parsing. In *ICON-2010 Tools Contest on Indian Language Dependency Parsing. Kharagpur, India.*

Riedel, Sebastian, Ruket Çakici, and Ivan Meza-Ruiz. 2006. Multi-lingual dependency parsing with incremental integer linear programming. In *CoNLL-X '06: Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 226–230, Morristown, NJ, USA. Association for Computational Linguistics.

Robinson, Jane J. 1970. Dependency structures and transformational rules. *Language*, 46:259–285.

Sagae, Kenji, Eric Davis, Alon Lavie, Brian MacWhinney, and Shuly Wintner. 2007. High-accuracy annotation and parsing of CHILDES transcripts. In *CACLA '07: Proceedings of the Workshop on Cognitive Aspects of Computational Language Acquisition*, pages 25–32, Morristown, NJ, USA. Association for Computational Linguistics.

Sagae, Kenji and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, NAACL-Short '06, pages 129–132, Stroudsburg, PA, USA. Association for Computational Linguistics.

Sarafraz, Farzaneh and Goran Nenadic. 2010. Identification of negated regulation events in the literature: exploring the feature space. In *Semantic Mining in Biomedicine.*

Schiehlen, Michael and Kristina Spranger. 2006. Language independent probabilistic context-free parsing bolstered by machine learning. In *CoNLL-X '06: Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 231–235, Morristown, NJ, USA. Association for Computational Linguistics.

Shimizu, Nobuyuki. 2006. Maximum spanning tree algorithm for non-projective labeled dependency parsing. In *CoNLL-X '06: Proceedings*

*of the Tenth Conference on Computational Natural Language Learning*, pages 236–240, Morristown, NJ, USA. Association for Computational Linguistics.

Shinyama, Yusuke, Satoshi Sekine, and Kiyoshi Sudo. 2002. Automatic paraphrase acquisition from news articles. In *Proceedings of the second international conference on Human Language Technology Research*, pages 313–318, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Siddharthan, Advaith. 2002. Resolving attachment and clause boundary amgiguities for simplifying relative clause constructs. In *Proceedings of the Student Research Workshop, 40th Meeting of the Association for Computacional Linguistics.*

Siddharthan, Advaith. 2003. *Syntactic Simplification and Text Cohesion.* Ph.D. thesis, Research on Language and Computation.

Simov, Kiril, Petya Osenova, Alexander Simov, and Milen Kouylekov. 2005. Design and implementation of the Bulgarian HPSG-based treebank. *Journal of Research on Language and Computation – Special Issue*, 2(4):495–522, December.

Simov, Kiril, Petya Osenova, and Milena Slavcheva. 2004. BulTreeBank morphosyntactic tagset. Technical Report BTB-TR03, BulTreeBank Project, March.

Simov, Kiril, Petya Osenova, Milena Slavcheva, Sia Kolkovska, Elisaveta Balabanova, Dimitar Doikoff, Krassimira Ivanova, Alexander Simov, Er Simov, and Milen Kouylekov. 2002. Building a linguistically interpreted corpus of bulgarian: the bultreebank. In *In: Proceedings of LREC 2002, Canary Islands*, page pages.

Simov, Kiril, Gergana Popova, and Petya Osenova. 2002. HPSG-based syntactic treebank of Bulgarian (BulTreeBank). pages 135–142.

Smrž, Otakar, Jan Šnaidauf, and Petr Zemánek. 2002. Prague dependency treebank for Arabic: Multi-level annotation of Arabic corpus. pages 147–155.

Snow, Catherine E., United States., Science, and Technology Policy Institute (Rand Corporation). 2002. *Reading for understanding : toward an R&D program in reading comprehension / Catherine Snow.* Rand, Santa Monica, CA :.

Snow, Rion, Daniel Jurafsky, and Andrew Y Ng. 2005. Learning Syntactic Patterns for Automatic Hypernym Discovery. In *Advances in Neural Information Processing Systems 17*, Cambridge, MA.

Surdeanu, Mihai, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *CoNLL 2008: Proceedings of the 12th Conference on Natural Language Learning*, pages 159–177, Manchester, United Kingdom.

Szarvas, György, Veronika Vincze, Richárd Farkas, and János Csirik. 2008. The bioscope corpus: annotation for negation, uncertainty and their scope in biomedical texts. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing*, BioNLP '08, pages 38–45, Stroudsburg, PA, USA. Association for Computational Linguistics.

Tesnière, Lucien. 1959. *Éléments de syntaxe structurale*. Editions Klincksieck.

Tiedemann, Jörg. 2012. Parallel data, tools and interfaces in opus. In Nicoletta Calzolari (Conference Chair), editor, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, may. European Language Resources Association (ELRA).

Tjong Kim Sang, Erik F. and Hervé Déjean. 2001. Introduction to the conll-2001 shared task: Clause identification. In Walter Daelemans and Rémi Zajac, editors, *Proceedings of CoNLL-2001*, pages 53–57. Toulouse, France.

Tsarfaty, Reut, Joakim Nivre, and Evelina Andersson. 2011. Evaluating dependency parsing: Robust and heuristics-free cross-annotation evaluation. In *EMNLP*, pages 385–396.

Tsarfaty, Reut, Joakim Nivre, and Evelina Andersson. 2012. Cross-framework evaluation for statistical parsing. In *EACL*, pages 44–54.

van den Bosch, Antal. 2004. Wrapped progressive sampling search for optimizing learning algorithm parameters. In *Proceedings of the 16th Belgian-Dutch Conference on Artificial Intelligence*.

Van der Beek, Leonoor, Gosse Bouma, Jan Daciuk, Tanja Gaustad, Robert Malouf, Gertjan van Noord, Robbert Prins, and Begoña Villada. 2002a. The Alpino dependency treebank. In *Algorithms for Linguistic Processing*. NWO PIONIER progress report 5.

Van der Beek, Leonoor, Gosse Bouma, Robert Malouf, and Gertjan van Noord. 2002b. The Alpino dependency treebank. In *Computational Linguistics in the Netherlands (CLIN)*. Rodopi.

Vincze, Veronika, Gyorgy Szarvas, Richard Farkas, Gyorgy Mora, and Janos Csirik. 2008. The BioScope corpus: biomedical texts annotated for uncertainty, negation and their scopes. *BMC Bioinformatics*, 9(Suppl 11):S9+.

Williams, Sandra, Ehud Reiter, and Liesl M. Osman. 2003. Experiments with discourse-level choices and readability. In *In Proceedings of the European Natural Language Generation Workshop (ENLG) and 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL03)*, pages 127–134.

Wu, Yu-Chieh, Yue-Shi Lee, and Jie-Chi Yang. 2006. The exploration of deterministic and efficient dependency parsing. In *CoNLL-X '06: Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 241–245, Morristown, NJ, USA. Association for Computational Linguistics.

Yamada, H. and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of International Workshop of Parsing Technologies (IWPT'03)*, pages 195–206.

Yli-Jyrä, Anssi Mikael. 2003. Multiplanarity – a model for dependency structures in treebanks. In Joakim Nivre and Erhard Hinrichs, editors, *TLT 2003. Proceedings of the Second Workshop on Treebanks and Linguistic Theories*, volume 9 of *Mathematical Modelling in Physics, Engineering and Cognitive Sciences*, pages 189–200, Växjö, Sweden, 14-15 November. Växjö University Press.

Yuret, Deniz. 2006. Dependency parsing as a classification problem. In *CoNLL-X '06: Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 246–250, Morristown, NJ, USA. Association for Computational Linguistics.

Zeman, Daniel and Zden K Zabokrtský. 2005. Improving Parsing Accuracy by Combining Diverse Dependency Parsers. In *Proceedings of the 9th International Workshop on Parsing Technologies*.

Zhu, Qiaoming, Junhui Li, Hongling Wang, and Guodong Zhou. 2010. A unified framework for scope learning via simplified shallow semantic parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 714–724, Stroudsburg, PA, USA. Association for Computational Linguistics.